Rogério Alexandre Botelho Campos Rebelo

Geração automática de código ANSI C a partir de Redes de Petri IOPT

- PNML2C -

Lisboa

2010

UNIVERSIDADE NOVA DE LISBOA

Faculdade de Ciências e Tecnologia

Depº de Engenharia Electrotécnica

Geração automática de código ANSI C a partir de Redes de Petri IOPT

- PNML2C -

Por:

Rogério Alexandre Botelho Campos Rebelo

Dissertação apresentada na Faculdade de Ciências e Tecnologia da Universidade Nova de Lisboa para obtenção do Grau de Mestre em Engenharia Electrotécnica e de Computadores

Orientador: Prof. Doutor Luís Filipe dos Santos Gomes

Lisboa

2010

AGRADECIMENTOS

Ao Prof. Doutor Luís Gomes por todo o apoio, orientação e disponibilidade que tornaram esta dissertação possível.

Aos meus pais, Saul Rebelo e Filomena Botelho e irmã, Catarina Rebelo, pela paciência e atenção que me prestaram quando necessário.

Aos meus amigos e colegas pelo apoio e força que me foram dando tanto durante esta dissertação como ao longo de todo o curso. Um obrigado especial ao João Simões e ao Bruno Rodrigues pelo apoio e pelos momentos que passámos, neste último ano em especial.

A todos, que de uma maneira ou de outra me deram força para continuar.

SUMÁRIO

Nesta dissertação apresenta-se a elaboração de regras de tradução e o desenvolvimento de uma ferramenta de geração de código na Linguagem ANSI C a partir de Redes de Petri IOPT especificadas em PNML. As redes de Petri IOPT resultam da extensão das redes de Petri Lugar-Transição com algumas características não-autónomas, como sinais e eventos de entrada e saída.

Para cada característica dos modelos expressos em Redes de Petri IOPT são definidas as estruturas necessárias em ANSI C para a sua execução, de forma a garantir a semântica pretendida.

A ferramenta desenvolvida recebe o modelo do controlador expresso através de um modelo IOPT através de um ficheiro no formato PNML, ao qual foram adicionadas as representações dos elementos específicos das redes IOPT. O modelo é analisado e é criado um conjunto de cinco ficheiros preparados para, apenas com a definição do interface com a plataforma de implementação, executar a rede de acordo com a semântica definida.

Foram criados dois interfaces para a ferramenta, um gráfico para ser usado por humanos e outro por comandos para facilitar a integração com outras ferramentas.

Criou-se ainda uma nova ferramenta com o mesmo objectivo mas utilizando um novo meta-modelo das Redes de Petri IOPT de forma a tratar as alterações introduzidas por este meta-modelo. As ferramentas desenvolvidas foram validadas através de exemplos de sistemas de automação

ABSTRACT

In this dissertation it's presented the elaboration of rules of translation and the development of a tool for code generation in ANSI C language through IOPT Petri Nets specified in PNML. IOPT Petri Nets are a result of the extension of place-transition Petri Nets with some non autonomous characteristics, like signals and events of output and input.

For each characteristic of the models expressed in IOPT Petri Nets, structures necessary in ANSI C are defined for their execution, guarantying the pretended semantics.

The developed tool receives the express controller model through an IOPT model through a file in PNML format, to which representations of specific elements of IOPT nets were added. The model is analyzed and a set of five files is created, these are prepared so that with only the definition of the interface with the implementation platform, they can execute the net according to the defined semantics.

Two interfaces were created for the tool, a graph to be used by humans and another through commands to facilitate the integration with other tools.

A new tool was created with the same objective but using a new IOPT Petri Nets meta-model so that the alterations introduced by this meta-model are treated. The developed tools were validated through examples of automated systems.

SIMBOLOGIA E NOTAÇÕES

FORDESIGN	Formal Methods for Embedded Systems co-Design
HTML	Hypertext Markup Language
IOPT	Input-Output Place Transition
ISO	International Standards Organization
PNML	Petri Net Markup Language
RdP	Redes de Petri (Petri Nets)
SGML	Standard Generalized Markup Language
WEB	World Wide Web
XML	Extensible Hypertext Markup Language

ÍNDICE DE MATÉRIAS

ÍNDICE DE FIGURAS

ÍNDICE DE QUADROS

1. INTRODUÇÃO

Neste capítulo pretende-se apresentar a motivação que levou à realização deste trabalho, tal como os seus objectivos e a sua estrutura.

1.1. MOTIVAÇÃO

Devido à sua flexibilidade, formalismo e representação gráfica que lhe permitem modelar problemas complexos, as Redes de Petri (RdP) têm vindo a ser usadas cada vez mais na modelação de sistemas nas mais diversas áreas, principalmente nas áreas da engenharia.

Devido a este crescimento, a procura por ferramentas que possibilitem a utilização mais eficiente das Redes de Petri é actualmente uma realidade que cresce de dia para dia.

Esta necessidade de ferramentas para a utilização de Redes de Petri está a impulsionar o aparecimento de projectos como o Fordesign [Fordesign, 07] onde são criadas ferramentas para a utilização deste formalismo tais como, geradores automáticos de código ou ferramentas de execução e simulação de Redes de Petri, do qual falaremos mais à frente nesta dissertação.

Esta tese vem no seguimento deste projecto tentando dar mais um passo na criação de ferramentas de geração automática de código, neste caso para ANSI C.

1.2. OBJECTIVOS

Esta Dissertação tem como objectivo a realização de uma ferramenta de geração automática de código ANSI C a partir de sistemas modelados em Redes de Petri IOPT [Gomes et al, 07], bem como a definição das regras de tradução necessárias para esta geração.

O gerador deverá partir de um ficheiro com as especificações de Redes de Petri IOPT descrito em Petri Net Markup Language (PNML) [PNML, 04] e aplicando as regras de tradução gerar um conjunto de ficheiros em ANSI C que permitam a execução da rede.

O conjunto de ficheiros gerados deverá ser robusto o suficiente para permitir uma execução eficaz em diferentes plataformas, mediante a definição pelo utilizador do interface com a plataforma onde o sistema será implementado.

Deve também definir-se um conjunto de regras de tradução que permita associar cada parte constituinte da Rede de Petri à forma de a representar em ANSI C.

2. FORMALISMOS, LINGUAGENS E CONCEITOS DE INTERESSE

Neste capitulo serão apresentados alguns conceitos como formalismos de modelação e linguagem de programação importantes para o desenvolvimento desta dissertação.

2.1. FORMALISMOS DE MODELAÇÃO

A fase de modelação e especificação de um sistema é em todos os ramos da Engenharia bastante relevante, principalmente em sistemas de maior complexidade.

Tendo em conta que os sistemas embutidos estão a tornar-se cada vez mais complexos devido a factores como o aumento da capacidade dos dispositivos, permitindo o aumento da dimensão dos sistemas e o aumento da sua fiabilidade, as restrições ao consumo de energia, os custos e o tempo de desenvolvimento (time – to - market), esta fase torna-se fundamental na realização destes.

A realização desta fase de modelação permite aos modeladores de sistemas embutidos uma melhoria na percepção do comportamento do sistema. Dá-lhes também a possibilidade de verificação e validação do seu funcionamento.

A utilização destes modelos permite também a utilização de várias ferramentas baseadas no mesmo modelo, permitindo a realização do sistema por várias pessoas e até em ferramentas diferentes. [Sgroi, 00]

Desta forma a modelação de um sistema trás muitas vantagens pois permite uma compreensão mais completa do sistema, tal como a sua validação, viabilidade e custos de desenvolvimento e implementação.

Para melhorar a fase de Modelação, têm sido introduzidos vários formalismos de modelação, baseados em modelos matemáticos, que se propõem especificar o comportamento dos sistemas de uma forma precisa e o mais realista possível.

Estes formalismos de modelação ou Modelos de Computação são constituídos por uma notação e por regras de computação do comportamento [Gomes et al, 06].

Em casos de modelos bastante complexos e heterogéneos pode ser mais eficiente modelar o sistema com mais do que um formalismo. Nesse caso deve decompor-se o modelo em vários submodelos aplicando assim o formalismo mais indicado a cada um.

Nas secções seguintes serão apresentados dois exemplos destes formalismos, os fluxogramas como o mais antigo e utilizado e as redes de Petri, utilizadas nesta dissertação.

2.1.1. FLUXOGRAMAS

Os fluxogramas são provavelmente o formalismo de modelação mais antigo e mais utilizado por modeladores de sistemas, em todas as áreas, para representar e esquematizar sistemas.

São provavelmente o formalismo mais simples o que trás grandes vantagens, mas também o mais limitado para modelar sistemas mais complexos.

Apesar de os fluxogramas terem aparecido num trabalho da universidade de Princeton por volta de 1947, apenas 41 anos depois e após várias evoluções ao modelo inicial foi definida uma norma, o ISO 5807, 1985.

Em seguida apresenta-se resumidamente a notação gráfica usada por este formalismo:

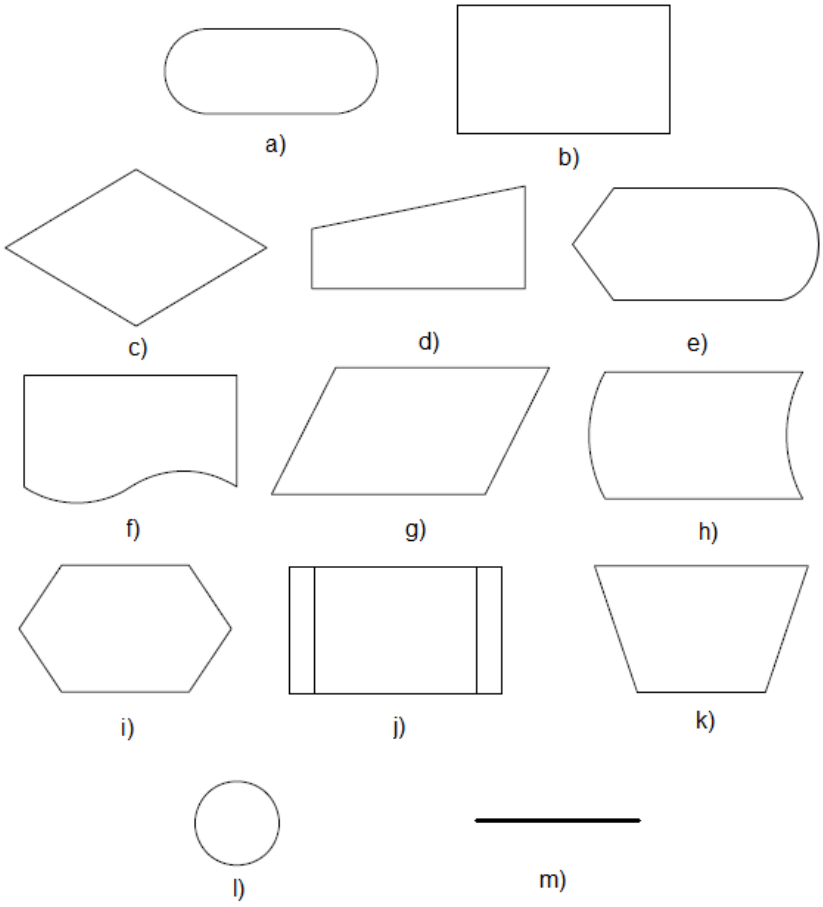

Figura 2.1 – Notação gráfica de um fluxograma.
a) Terminal; b) Processo;
c) Decisão; d) Entrada Manual; e) Exibição;
f) Documento; g) Dados; h) Dados gravados;
i) Preparação; j) Processo Predefinido; k) Operação manual;
l) Conector; m) Linha.

Terminal

O terminal representa o inicio e o fim do fluxo do programa. Pode ainda ser usado na definição de subrotinas. É representado como na figura 2.1 a).

Processamento

A figura 2.1 b) mostra a representação gráfica de um processamento. Este representa a execução de uma operação ou grupo de operações que estabelecem de um cálculo.

Decisão

A decisão representa uma condição que desvia o fluxo para outros pontos dependendo da resposta a esta condição. Dependendo da resposta o fluxo segue caminhos diferentes. A forma de a representar está representada na figura 2.1. c).

Entrada Manual

Representa a introdução manual de dados por um utilizador humano por meio de um interface como por exemplo o teclado. A forma de representar é mostrada na figura 2.1. d).

Exibição

Representa-se como mostra a figura 2.1 e) e caracteriza a execução da operação de saída visual de dados num qualquer tipo de saída visual para o utilizador como por exemplo um monitor.

Documento

Representa-se como mostra a figura 2.1 f) e caracteriza a saída de dados num documento como por exemplo numa impressora.

Dados

Representa-se da forma apresentada na figura 2.1 g) e é uma descrição de dados de forma genérica. Tipicamente é associado a operações genéricas de entrada e saída de dados.

Dados gravados

Representa-se como na figura 2.1 h) que caracteriza o acesso a um ficheiro guardado, seja para leitura ou para escrita.

Preparação

Representa a modificação de instruções existentes em relação à actividade subsequêncial e é representado como na 2.1 i).

Processo Predefinido

Representa-se como na figura 2.1 j) e caracteriza a definição de um grupo de operações estabelecidas como uma subrotina de processamento anexo ao diagrama de blocos.

Operação Manual

Representa-se como na figura 2.1 k) e descreve a execução de qualquer operação que possa ser realizada pelo utilizador do sistema.

Conector

Representa-se por um círculo como na figura 2.1 l) e descreve a entrada e saída em partes do diagrama de blocos. Pode ser usado na definição de quebras de linha e na continuação da execução de decisões.

Linha

Representa-se por uma linha como apresentado na figura 2.1 m) e caracteriza as relações existentes entre os vários símbolos do diagrama. Normalmente tem a ponta em forma de seta indicando a direcção do fluxo de acção [Manzano, 04].

Na figura 2.2 apresenta-se um exemplo de um fluxograma para o cálculo da função factorial F = N!

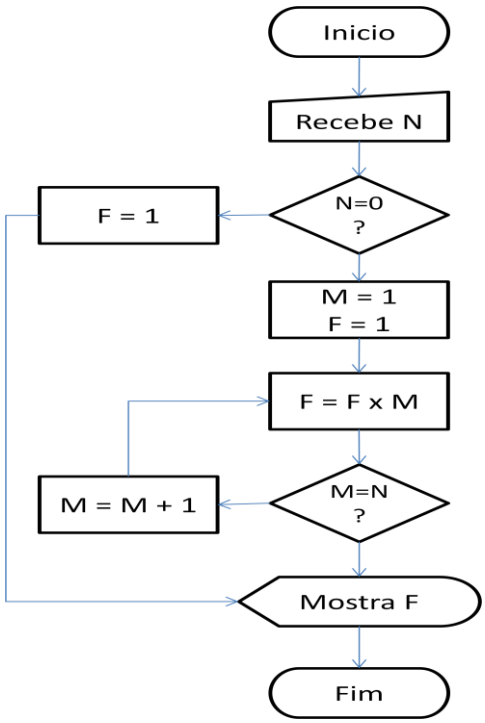

Figura 2.2 - Exemplo de um fluxograma.

O programa inicia e recebe do utilizador o valor N. Em seguida vai verificar se o valor de N é zero. Nesse caso o valor de F é definido com o valor um e é mostrado ao utilizador. Caso o valor de N seja diferente de zero inicializa-se M e F com o valor um. Em seguida multiplica-se F por M e coloca-se em F. Verifica-se se M é igual a N. Caso não seja adiciona-se um a M e volta-se a multiplicar a F colocando em F. Quando o valor de M for igual a N apresenta-se o valor de F no ecrã e termina o programa.

2.1.2. REDES DE PETRI

2.1.2.1. HISTÓRIA DAS REDES DE PETRI

As Redes de Petri foram criadas por Carl Adam Petri em 1962 No âmbito da sua dissertação de Doutoramento na Faculdade de Matemática e Física da Universidade Técnica de Darmstadt na Alemanha [Petri, 62] [Petri Nets World, 2010].

Nesta dissertação Petri apresentou uma variedade de ideias e propostas acerca dos fundamentos da Informática.

No ano seguinte Petri prova que a sua arquitectura é computacionalmente universal [Petri, 63].

Ao longo dos seus estudos Petri utiliza notações formais para sistemas distribuídos assíncronos compostas por representações gráficas de fórmulas algébricas com o operador “paralelo”. Esta Representação gráfica veio a resultar nas Redes de Petri, com lugares e transições [Brauer-Reisig, 06].

As redes de Petri como são conhecidas actualmente foram apresentadas por Petri em 1966 num Colóquio em Hannover [Petri, 67].

2.1.2.2. CONCEITO DE REDES DE PETRI

As grandes vantagens apresentadas pelas redes de Petri são a semântica e sintaxe exactas, a modelação explícita de características como o paralelismo, a sincronização e a gestão de recursos, a par de uma boa legibilidade dos modelos produzidos e do

suporte para as abordagens associadas à composição ascendente de modelos (“bottom-up”) e à decomposição descendente (“top-down”).

Apesar de apenas terem sido apresentadas em 1966, as Redes de Petri eram já em 1964 usadas na Bells Telephone Laboratories [Brauer-Reisig, 06] e em 1968 são aplicadas também no projecto Information System Theory da Applied Data Research, Inc, o que mostra a sua aplicabilidade na modelação de sistemas concorrentes [Heuser, 91].

As Redes de Petri permitem também a visualização simultânea da estrutura e do comportamento do sistema [Zurawski et al, 94], tendo ainda a capacidade de modelar problemas como o sincronismo, concorrência, conflitos e partilha de recursos [Peterson, 77].

Com as Redes de Petri é possível capturar a dinâmica dos sistemas tornando-se assim particularmente úteis para simulação [Murata, 89].

2.1.2.3. ESTRUTURA DAS REDES DE PETRI

A estrutura de uma rede de Petri pode ser definida como um tuplo R = (L, T, AE, AS) [Peterson, 77] como se mostra na Tabela 2.1.

R = (L, T, AE, AS) onde:
L = { p1, p2, ..., pm} é um conjunto de lugares
T = { t1, t2, ..., tn} é um conjunto de transições
L ∩ T = ∅ os conjuntos L e T são disjuntos
AE: L x T é um conjunto de arcos de entrada nas transições
AS: T x L é um conjunto de arcos de saída das transições

Por exemplo, para a figura 2.4 temos:
L = { p0, p1, p2, p3, p4}
T = { t0, t1, t2, t3 }
AE = { (p0, t0), (p1, t1), (p2, t2), (p3, t3), (p4, t3) }
AS = { (t0, p1), (t 0, p2), (t1, p3), (t2, p4), (t3, p0) }

E ainda um conjunto de marcações dos lugares:

M = (0,1,0,0,1)

Tabela 2.1 - Definição de uma Rede de Petri [Peterson, 77], adaptado de [Barros, 96].

A representação gráfica das RdP é feita através de um grafo com dois tipos de nós: lugares e transições ligados através de arcos. Estes são os três elementos constituintes de uma RdP [Murata, 89].

Apesar de esta ser a definição mais aceite, existem alguns autores que consideram ainda a marca o quarto elemento constituinte das RdP [Moen, 03].

Em seguida apresentam-se estes três elementos:

O lugar está associado a um estado do sistema e modela o comportamento estático do sistema. Contêm marcas que determinam a dinâmica do sistema e são representadas por pontos. Os Lugares são representados por circunferências ou elipses como mostra a figura 2.3 a). [Barros, 96] [Pais, 04].

A transição encontra-se entre lugares e é responsável pela criação e destruição de marcas nos lugares através do seu disparo. Desta forma é também responsável pela evolução do estado do sistema. São representadas por segmentos de recta, rectângulos ou barras como se mostra na figura 2.3 b) [Barros, 96].

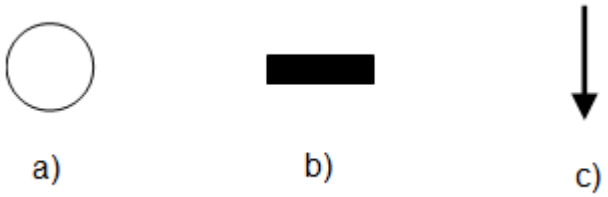

Figura 2.3 - Elementos de uma Rede de Petri.

a) Lugar; b) Transição; c) Arco.

O arco é o elemento que liga lugares a transições e transições a lugares. Ao arco está associado um peso que define quantas marcas vão ser transportadas. Aos arcos que ligam um lugar a uma transição pode chamar-se arco de entrada e o seu peso define as marcas necessárias no lugar para que a transição dispare. Aos arcos que ligam uma transição a um lugar pode chamar-se arco de saída e o seu peso define o número de marcas que vão ser criadas no lugar quando a transição dispara. Um arco nunca pode estar a ligar dois lugares ou duas transições. São representados por setas como se apresenta na figura 2.3 c) [Conceição, 04] [Barros, 96].

Com este conjunto de elementos é possível criar uma rede como a da figura 2.4. A RdP apresentada é designada por RdP marcada pois os lugares estão marcados.

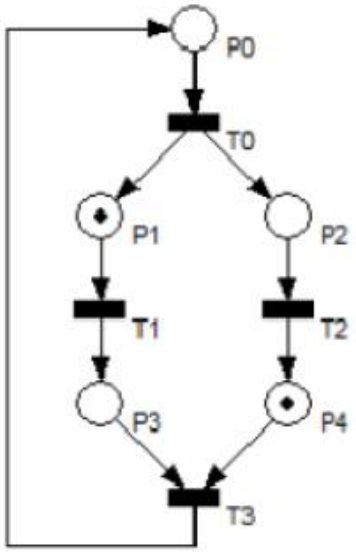

Figura 2.4 - Exemplo de uma RdP marcada, adaptado de [Reis, 08].

2.1.2.4. DISPARO DAS TRANSIÇÕES

Conforme foi referido anteriormente são as transições que podem destruir e criar as marcas contidas nos lugares. Os arcos permitem indicar sobre que lugares cada transição actua. Isto acontece quando a transição dispara. Para isso acontecer ela tem de estar habilitada.

Para uma transição estar habilitada têm de existir em todos lugares de entrada um numero de marcas igual ou superior ao peso do arco que o liga à transição [Barros, 96].

2.1.2.5. MODELAÇÃO COM REDES DE PETRI

As RdP têm a capacidade de modelar e permitir visualizar vários conceitos [Lino, 03]:

O **paralelismo/concorrência**, onde mais que um processo são executados em simultâneo como apresentado na figura 2.5 a).

O **conflito**, que ocorre quando como na figura 2.5 b) tanto a transição t1 como a t2 estão habilitadas mas o disparo de uma desabilita a outra, tendo de se decidir qual delas irá disparar.

A **confusão**, que ocorre quando como na figura 2.5 c) se a transição t1 disparar antes da t3 acontece uma situação de conflito, já referida anteriormente.

A **sincronização**, onde se pode sincronizar vários processos com tempos de execução diferentes de modo a que antes de começar outro, todos eles tenham acabado. Como na figura 2.5 d), onde a transição t1 só fica habilitada quando os dois lugares estão marcados, ou seja, quando os dois processos que acabam naqueles lugares estão terminados.

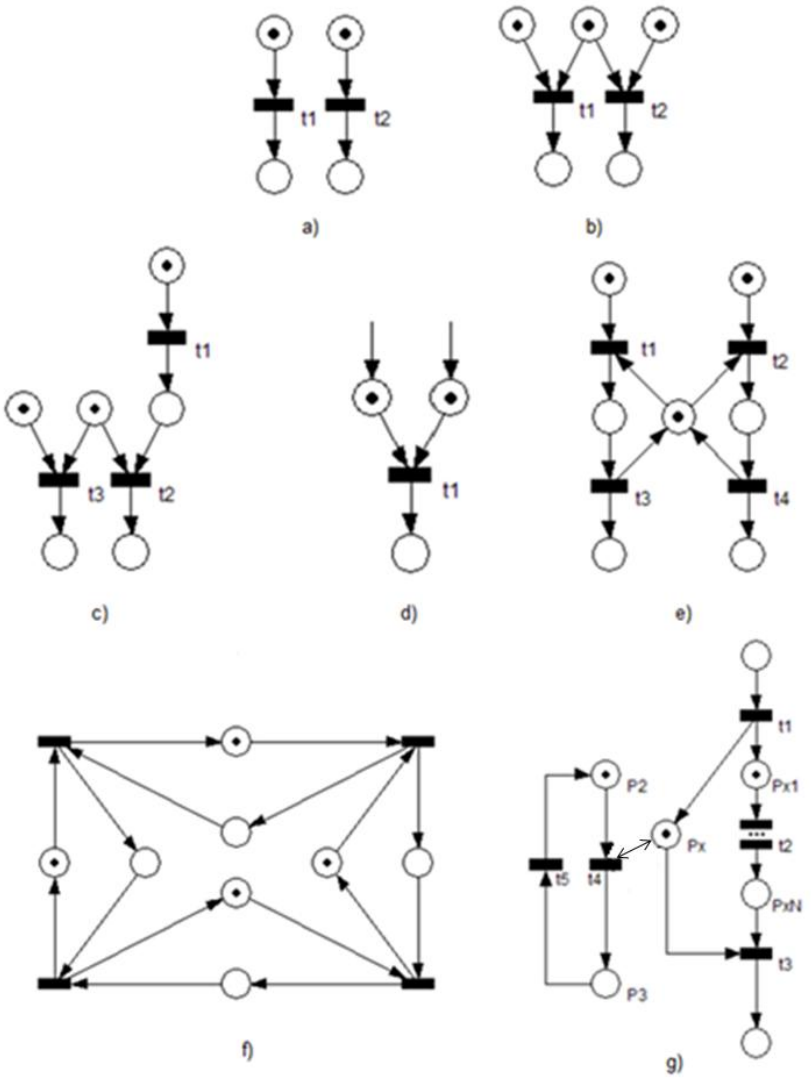

Figura 2.5 - Problemas que as RdP permitem Modelar.

a) Paralelismo; b) Conflito;
c) Confusão; d) Sincronização; e) Gestão de Recursos;
f) Lugares Complementares; g) Arco de teste.

A **gestão de recursos**, que permite a gestão de um recurso critico, que por exemplo só pode ser usado por um processo de cada vez. Na figura 2.5 e) tanto a transição t1 como a t2 precisam de uma marca do mesmo lugar, logo apesar de poderem estar as

duas habilitadas ao mesmo tempo, apenas uma pode disparar deixando a outra de estar habilitada. Desta forma, apenas um processo usa o recurso de cada vez.

Os **lugares complementares**, que como se mostra na figura 2.5 f), têm sempre o complemento da marcação do lugar a que estão associados.

O **teste à marcação** que é feito por arcos de teste. O arco de teste não consome marcas do lugar a que está associado. Apenas verifica se elas existem. Na figura 2.5 g) podemos, usando o mesmo exemplo mas desta vez com um arco de teste verificar se o arco Px está marcado. A marcação de Px habilita a transição t4 mas não é destruída, Mantendo-se o lugar Px marcado.

2.1.2.6. PNML

Na década de 1970, devido à grande difusão da tecnologia de informação, foi necessário criar normas que permitissem a elaboração de documentos de forma a que permanecessem legíveis por muito tempo.

Em 1986 surgiu a norma SGML (ISO 8879:1986 Information processing – text and Office systems – Standard Generalized Markup Language) [SGML, 04]. Esta é uma metalinguagem na qual se podem definir linguagens de “marcação” para documentos, fornecendo uma variedade de sintaxes de marcação que podem ser usadas para partilhar documentos entre várias máquinas ou aplicações.

A partir da norma SGML surgiram várias derivações, das quais se destacam o HTML [HTML, 10], o DocBook [DocBook, 10] e o XML [XML, 03].

O HTML é a linguagem de marcação para páginas Web que fornece um meio de descrever em forma de texto a estrutura da informação da página.

O DocBook foi concebido para a elaboração de documentação técnica relacionada com hardware e software.

O XML [XML, 03] é uma linguagem extensível que permite ao utilizador a definição das inscrições (tags) permitindo a criação de linguagens para comunicação de dados entre diferentes sistemas.

O PNML (Petri Net Markup Language) é uma linguagem de descrição de RdP baseada em XML (Extensible Markup Language) que especifica os diferentes atributos da RdP através de etiquetas (tags) tal como no HTML.

O PNML é apresentado como a norma de representação de RdP em [Jürgel et.al, 00] e [Weber-kindler, 02]. Apresentando-se como um formato único e universal permite uma maior portabilidade entre ferramentas e projectos.

O PNML apresenta-se com os seguintes princípios:

Legibilidade, sendo legível e editável por humanos através de um qualquer editor de texto. Isto é garantido pelo uso de XML.

Universalidade, não excluindo qualquer versão de RdP, permitindo a representação de qualquer tipo de extensão. Garantido com o uso de um arquivo denominado Petri Net Type Definition (PNTD) que define etiquetas, valores e combinações de valores autorizados.

Mutabilidade, pois deve permitir representar toda a informação possível de uma RdP, inclusivamente de tipos desconhecidos. Deve representar os princípios comuns das RdP [Weber-Kindler, 02]. O que permite esta mutabilidade é a convenção que define o conjunto de etiquetas permitidas.

O facto de os sistemas reais serem demasiado grandes para que o seu desenho caiba numa página levou a que as ferramentas de desenho permitam estruturar o sistema através de módulos ou páginas [Weber-Kindler, 02].

No PNML estruturado (Structured PNML) cada rede é uma página e cada página pode referenciar outras, permitindo o uso de redes de outras páginas.

No PNML modular (Modular PNML) cada rede é um módulo e pode ligar-se a outros módulos através de nós [Kindler -Weber, 01] [Weber-Kindler, 03].

2.1.2.7. CLASSES DE REDES DE PETRI

Com a evolução das RdP têm aparecido várias variações ao modelo inicial, de forma a permitir o uso mais eficiente das RdP em novas áreas. A estas variações dá-se o nome de classes e alguns exemplos são as RdP Coloridas, as RdP Reactivas (RdP-R), as RdP Reactivas e Hierárquicas (RdP-RH) ou as RdP de controlo interpretado por cor (Coloured Control Interpreted Petri Nets).[Barros,96] [Wegrzyn et al, 97] [Gomes, 97] [Jensen, 97].

Pode Classificar-se as RdP segundo dois critérios. Em relação à dependência das características físicas, alterando as regras de disparo das transições podem classificar-se como RdP autónomas e RdP não-autónomas. Em relação ao tipo de marcas, se são todas iguais ou não pode classificar-se as RdP como Baixo Nível ou Alto Nível.

Na Tabela 2.2 podem ver-se alguns exemplos de classes de RdP classificadas segundo estes dois critérios.

RdP	**Não-Autónomas** estão directamente ligadas à modelação de sistemas físicos	**Autónomas** as regras de disparo só dependem das caracteristicas do grafo.
Baixo Nível são caracterizadas por lugares marcados em que as marcas são indistintas entre si	**Interpretadas Sincronizadas Temporizadas**	**Condição-evento Elementares Lugar-transição**
Alto Nível são caracterizadas por lugares que podem conter marcas com uma estrutura de dados associada	**Reactivas**	**Coloridas**

Tabela 2.2 - Classes de RdP. Adaptado de [Lino, 03].

2.1.2.7.1. IOPT (INPUT-OUTPUT PLACE-TRANSITION)

As RdP IOPT (Input-Output Place-Transition) são baseadas na classe Lugar-Transição. As IOPT, apresentadas em [Gomes et al, 07], estendem as RdP Lugar-Transição, permitindo a interacção com o exterior através de entradas e saídas (sinais e eventos). Estes sinais e eventos são associados ao disparo das transições e à marcação dos lugares, permitindo a comunicação com o exterior. Estas são redes de baixo nível e não-autónomas.

A extensão IO acrescenta um conjunto mínimo de notações às RdP Lugar-Transição que permitem a especificação de controlo de sistemas de eventos discretos. As IOPT permitem especificar:

- Sinais de entrada externos, eventos de entrada externos e eventos de saída externos, associados ao disparo das transições.
- Sinais de saída externos associados às marcações dos lugares.

Uma RdP IOPT é definida em [Gomes et al, 07] por uma sequência finita do tipo:

N = (L, T, A, AT, M, peso, peso teste, prioridade, gse, ee, es, css)

Em que:

- L é um conjunto finito de lugares.
- T é um conjunto finito de transições (disjunto de L).

- A é um conjunto de arcos, tal que $A \subseteq ((L \times T) \cup (T \times L))$.
- AT é um conjunto de arcos de teste, tal que $AT \subseteq (L \times T)$.
- M é a função de marcação: $M : L \rightarrow \mathbb{N}0$.
- Peso: $A \rightarrow \mathbb{N}0$.
- Peso teste: $AT \rightarrow \mathbb{N}0$.
- Prioridade é uma função parcial que aplica transições a inteiros não negativos: prioridade : $T \rightarrow \mathbb{N}0$.
- *gse* é uma função parcial de guardas do sinal de entrada que aplica transições a expressões Booleanas (onde todas as variáveis são sinais de entrada): *gse* : $T \rightarrow EB$, onde $\forall eb \in gse(T), Var(eb) \subseteq SE$.
- ee é uma função parcial de eventos de entrada que aplica transições para eventos de entrada. ee: $T \rightarrow EE$
- es é uma função parcial de eventos de saída que aplica transições para eventos de saída. es: $T \rightarrow ES$
- css é uma função de condições de sinais de saída dos lugares em conjuntos de regras: css: $P \rightarrow \mathcal{P}$ (*REGRAS*), quando $REGRAS \subseteq (BES \times SS \times \mathbb{N}\ 0)$, $BES \subseteq EB$ and $\forall e \in BES, Var(e) \subseteq ML$ com *ML* como o conjunto de identificadores de cada marcação de lugar após um passo de execução dado.

Os sinais de entrada externos podem ser associados a transições dentro de expressões de guarda.

Os sinais de saída podem ser modelados de duas formas:

a) Associados à marcação da rede, ou seja, às marcas/lugares, estilo Moore [Moore, 56];
b) Como sinais associados a eventos de saída produzidos aquando do disparo de uma transição em que o estado muda, estilo Mealy [Mealy, 55].

Quanto às regras de disparo das transições é adoptado o paradigma de sincronização [David-Alla, 92]. Significa que quando uma transição está habilitada, uma condição externa de evento associada a essa transição é verificada e a condição de guarda é verificada, ocorre o disparo.

Como comum na modelação de sistemas com eventos discretos, a evolução do modelo só é possível em instantes específicos, denominados “ticks”; o período entre dois “ticks” consecutivos é designado ciclo ou passo (step). Todas as transições que estejam habilitadas, disparam no mesmo passo(“maximal step”).

As RdP IOPT podem também conter macro-nós, por exemplo macro-lugares e macro-transições como mecanismos de estruturação [Gomes-Barros, 03], suportados por uma decomposição em super-páginas e sub-páginas [Jensen, 97]. A introdução destes conceitos é de extrema importância, do ponto de vista da Engenharia, para que as RdP deixem de ser classificadas como um método formal sem mecanismos de estruturação.

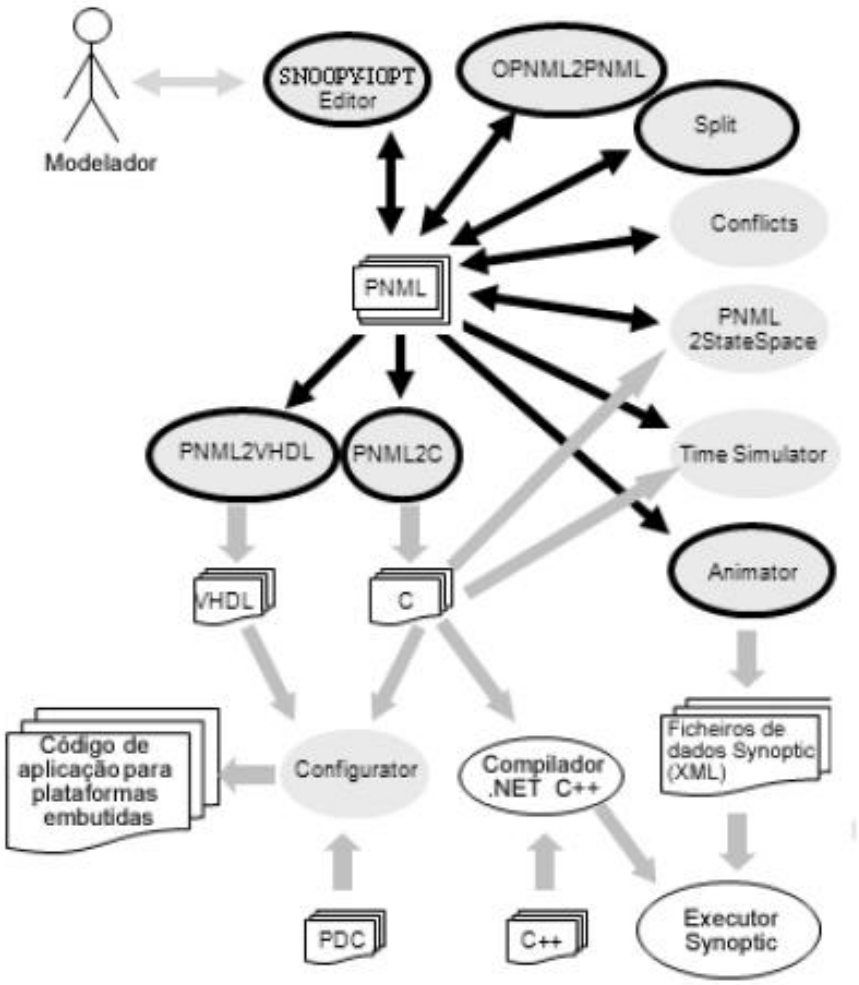

Figura 2.6 - Projecto Fordesign, adaptado de [Fordesign, 07].

No projecto FORDESIGN apresentado na Figura 2.6 foram desenvolvidas várias ferramentas para a utilização das redes IOPT. Alguns exemplos são o editor Snoopy IOPT que permite editar de forma gráfica RdP IOPT ou o Split que permite partir uma rede em várias sub-redes. Este projecto permite a utilização de IOPT facilitando a sua aplicação em plataformas físicas ou o teste e simulação destas.

2.2. LINGUAGEM C

A década de 1960 foi muito turbulenta para a pesquisa em sistemas de computadores na Bell Telephone Laboratories [Ritchie, 78] [Ritchie, 84]. O projecto Multics [Organick, 75], que tinha começado como uma empreendimento conjunto (joint venture) do MIT, General Electric e Bell Labs, em 1969, é abandonado pela empresa depois de ter chegado à conclusão que este seria demasiado moroso e dispendioso.

Ainda antes do fim do projecto já um grupo informal liderado a principio por Ken Thompson, tinha começado a investigar alternativas.

2.2.1. HISTÓRIA DA LINGUAGEM C

A linguagem C (derivada da linguagem sem tipos BCPL [Richards, 67]) foi criada por Dennis Ritchie entre 1969 e 1973, em paralelo com o desenvolvimento do sistema operativo UNIX, mas o ano de maior desenvolvimento foi 1972.

Outro grande pico de desenvolvimento foi entre 1977 e 1979, quando estava a ser demonstrada a portabilidade do sistema Unix. Durante este período apareceu a primeira descrição amplamente disponível da linguagem muitas vezes chamada livro branco (white book) ou “K&R”[Kernighan, 78]. No meio da década de 1980 a linguagem foi finalmente normalizada pelo ANSI X3J11 committee, fazendo algumas alterações.

Até à sua normalização, apesar de existirem compiladores para várias arquitecturas a linguagem era quase exclusivamente associada ao Unix. Mais recentemente o seu uso tornou-se muito mais amplo e actualmente é uma das linguagens mais usadas.

2.2.2. PORTABILIDADE

No início de 1973, os fundamentos de C modernos estavam completos. A linguagem e o compilador estavam robustos o suficiente para permitir reescrever o kernel do Unix para o PDP-11 em C. Também durante este período, o compilador foi redireccionado para outras máquinas próximas, particularmente o Honeywell 635 e IBM 360/370. Como a linguagem não pode viver isolada, os protótipos para as bibliotecas modernas foram desenvolvidos, em particular, Lesk escreveu um Pacote de I/O portável [Lesk, 73], que foi posteriormente reformulada para se tornar o padrão de rotinas I/O de C.

Durante 1973-1980, a língua cresceu um pouco. Adquiriu alguns tipos de estrutura como “unsigned” e “long”. Também nessa altura, como descrito em [Johnson, 78a], descobriu-se que os problemas que mais dificultavam a proliferação de ferramentas Unix não eram relacionados com a interacção da linguagem C com o novo hardware, mas com a adaptação do software existente de outros sistemas operativos. Assim, Steve Johnson começou a trabalhar num compilador C destinado a facilitar a reorientação para novas máquinas [Johnson 78b].

Durante a década de 1980 o uso da linguagem C espalhou-se, existindo compiladores disponíveis para quase todos os sistemas operativos. Tornou-se popular, em particular, como um instrumento de programação para computadores pessoais, tanto para os fabricantes de software comercial como para utilizadores finais interessados em programação.

2.2.3. NORMALIZAÇÃO

Em 1982 ficou claro que o C necessitava de uma norma formal. A melhor aproximação a uma norma, a primeira edição do K & R, já não descrevia a linguagem em uso na altura e era também demasiado precisa em muitos detalhes da linguagem, e tornou-se cada vez mais impraticável. Finalmente, o uso de C em projectos comerciais e governamentais tornou necessária a criação de uma norma oficial. Assim, a pedido de M. D. McIlroy, o ANSI (American National Standard Institute) estabeleceu em 1983 o comité X3J11 sob a direcção da CBEMA, com o objectivo de produzir um padrão de C.

O X3J11 produziu um relatório [ANSI, 89] no final de 1989 e, posteriormente, este padrão foi aceite pela ISO como ISO/IEC 9899-1990.

2.3. GERAÇÃO AUTOMÁTICA DE CÓDIGO

Para que uma tecnologia cresça ela tem de chegar ao maior número de pessoas. Com a sua evolução há necessidade de desenvolver ferramentas que permitam a quem a domina poder fazer cada vez mais coisas com ela.

Os geradores automáticos de código vêm desta forma permitir que o utilizador possa desenvolver projectos em várias linguagens utilizando para isso sempre a mesmo modelo, representado num determinado formalismo.

Como podemos verificar por uma análise das ferramentas utilizadas actualmente podem encontrar-se muitas onde é possível criar código utilizando especificações gráficas, principalmente para linguagens como HTML ou PHP. Alguns exemplos disso são ferramentas como o Macromedia Dreamweaver ou Joomla.

Uma pesquisa pela internet permite encontrar algumas ferramentas que permitem a modelação ou simulação de RdP tanto de âmbito comercial como de âmbito académico. Um exemplo é a ferramenta PEP [PEP, 10] que disponibiliza um ambiente gráfico onde é possível modelar, compilar, simular e verificar componentes. Esta ferramenta pode ser encontrada no site [Petri Nets World, 10].

Ferramentas de geração automática de código através de RdP não são ainda muito frequentes.

A ferramenta Petrify [Petrify, 10] é um exemplo de ferramentas de tradução de RdP para hardware através de circuitos assíncronos. Esta recebe uma RdP como entrada e a informação sobre os sinais e produz a informação para a criação de um controlador assíncrono.

Na área dos PLC's existe a ferramenta SIPN-Editor [SIPN, 10], desenvolvida exclusivamente para a implementação em PLC's que permite a edição, visualização e tradução hierárquica de RdP SIPN's, convertendo-as em instruções para implementar nesta tecnologia.

3. REGRAS DE TRADUÇÃO

Para realizar esta ferramenta foi necessário começar por definir regras de tradução de cada elemento de RdP para código C. Estas regras permitem especificar para cada elemento, objecto ou anotação da Rede IOPT a forma de os representar em código ANSI C de forma a permitir manter a semântica de execução pretendida.

Neste capítulo serão apresentadas as regras definidas para esta tradução, explicando para cada elemento da RdP IOPT a forma de o traduzir em código ANSI C.

3.1. EXECUÇÃO DE UMA ITERAÇÃO NA REDE

Ao falar numa iteração da rede fala-se em uma análise de todas as transições da RdP de forma a verificar quais as que se encontram habilitadas a disparar e fazendo-as disparar colocar a rede num novo estado de marcação com a criação e destruição de marcas realizadas pelo disparo das transições.

A figura 3.1 representa o fluxo de execução de uma iteração da rede. A execução começa por receber todos os sinais e eventos de entrada que são externos (input signals e input events) guardando-os nas variáveis definidas para esse efeito. Depois são analisados os sinais de entrada de forma a produzir os eventos de entrada dependentes da variação dos sinais. Em seguida o valor das marcações é guardada em variáveis auxiliares e as marcas geradas são inicializadas a zero. Após o qual são

analisadas as condições de disparo, disparando as condições habilitadas. Estes disparos criam marcas que são adicionadas nas variáveis de marcas geradas, retiram marcas da variável auxiliar que foram consumidas pelos disparos e criam sinais e eventos de saída (output signals e output events). Nesta fase são adicionadas as marcas geradas às marcas que estão na variável auxiliar e esse número de marcas é colocado no valor actual de marcas. Com esses valores de marcação é depois analisada a alteração de valores de sinais de saída. Também com os eventos de saída são efectuadas alterações aos sinais de saída. Finalmente os novos valores dos eventos de saída externos e dos sinais de saída ficam disponíveis para o exterior.

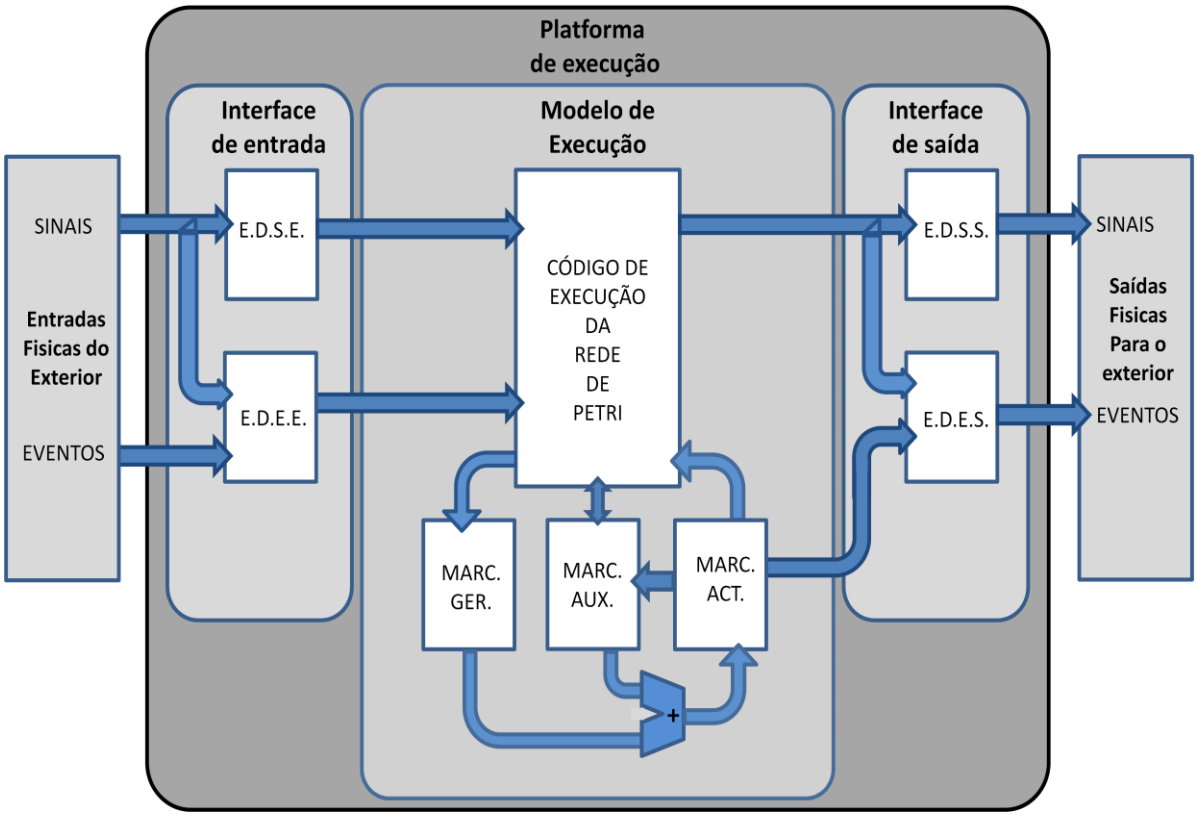

Figura 3.1 - Fluxo de execução de uma iteração na RdP. Adaptado de [Gomes et al, 10].

Na figura 3.1 as siglas e abreviaturas têm o seguinte significado:

E.D.S.E. – Estrutura de dados de sinais de entrada;

E.D.E.E. – Estrutura de dados de eventos de entrada;

E.D.S.S. – Estrutura de dados de sinais de saída;

E.D.E.S. – Estrutura de dados de eventos de saída;

MARC. GER. – Marcação gerada;

MARC. AUX. – Marcação auxiliar;

MARC. ACT. – Marcação actual.

A execução descrita está também representada nos passos seguintes:

1. *reset EDSE e EDEE*
2. *inicializar marcação actual*
3. *início ciclo de execução*
4. *recebe entradas pelo interface de entrada*
5. *reinicia EDSS e EDES*
6. *inicializar Marcação Auxiliar com a Marcação actual*
7. *reset a Marcação Gerada*
8. *disparo da rede e escrita de EDES*
9. *actualizar a marcação actual*
10. *actualizar a EDSS baseando-se na marcação actual*
11. *escrever saídas pelo interface de saída*
12. *fim ciclo de execução*

No disparo da rede o tratamento das transições é feito por ordem de prioridade. Desta forma a prioridade de cada transição é analisada e caso tenha o valor a ser tratado é adicionada a transição como será descrito mais à frente nesta dissertação.

A forma de ordenação das transições é descrita nos passos seguintes:

```
1.  Reset da prioridade actual
2.  Reset ao número de transições
3.  enquanto número de transições < Total Transições faz
4.      aumentar a Prioridade Actual
5.      para todas as transições faz
6.          se a Prioridade de Transição igual  a Prioridade Actual então
7.              (adicionar transição à lista de execução)
8.          fim se
9.      fim para
10. fim enquanto
```

Como se pode observar a lista de transições é percorrida procurando as transições com a prioridade a ser analisada, começando pela maior, o número de vezes necessárias até que todas as transições estejam tratadas.

3.2. LUGAR

Um lugar em RdP é representado em PNML como mostra a figura 3.2.

```
<place id="18061303">
  <name>
    <text>Car3_move_back</text>
    <graphics>
      <offset page="1" x="108" y="-3"/>
    </graphics>
  </name>
  <initialMarking>
    <graphics>
      <offset page="1" x="0" y="0"/>
    </graphics>
    <text>0</text>
  </initialMarking>
  <bound>
    <text>1</text>
  </bound>
  <signalOutputActions>
    <signalOutputAction idRef="outSignal_Car3MovesBack" value="1">
      <concreteSyntax language="VHDL">
        <text> </text>
      </concreteSyntax>
    </signalOutputAction>
    <signalOutputAction idRef="outSignal_Car3MovesBack" value="1">
      <concreteSyntax language="C">
        <text> </text>
      </concreteSyntax>
    </signalOutputAction>
  </signalOutputActions>
  <graphics>
    <position page="1" x="740" y="480"/>
  </graphics>
</place>
```

Figura 3.2 - Representação de um lugar em PNML.

Como se pode observar, retirando as características gráficas que apenas são necessárias para a representação gráfica da rede, restam como pontos importante o "*id"* e o nome do lugar, que identificam o lugar, o "*bound",* que representa o valor máximo de marcas que o lugar poderá ter durante a execução da rede, a marcação inicial do lugar e o conjunto de "*SignalOutputActions" associados ao lugar.*

O "*id"* e o nome são identificativos do lugar logo qualquer um deles poderia ser usado para identificar o lugar em C. Foi definido usar o "*id"* pois é gerado automaticamente e garantidamente único, ao contrário do nome que é dado pelo utilizador.

Os “*SignalOutputActions”* são tratados isoladamente no final da execução não sendo necessários na análise do disparo das transições, pois estão associados a sinais que podem ser alterados durante a execução.

Sobram duas características a ser tratadas. A marcação do lugar e o “*bound”.*

Então, um lugar de uma RdP corresponde em código C a um conjunto de 3 variáveis. Uma das variáveis tem guardado durante toda a iteração o valor de marcas no inicio desta e é declarada com o nome formado pelo “*id”* do lugar antecedido de “p_” e seguido de “_Mark”.

Numa outra variável são adicionadas as marcas criadas durante cada iteração. Esta variável é reiniciada no inicio de cada iteração e é declarada com o nome formado pelo “*id”* do lugar antecedido de “new_p_” e seguido de “_Mark”.

Finalmente a ultima variável que é reiniciada no inicio de cada iteração com o valor de marcas actual e onde são retiradas as marcas destruídas durante a iteração é declarada com o nome formado pelo “*id”* do lugar antecedido de “aux_p_” e seguido de “_Mark”.

Todas estas variáveis são declaradas com o mesmo tipo. O tipo de variável é definido pelo valor do *“bound”* podendo este ser *“char”*, *“int”* ou *“long”*.

```
char p_18061303_Mark;char aux_p_18061303_Mark;char new_p_18061303_Mark;
```

Figura 3.3 - Representação de um lugar em ANSI C.

Na figura 3.3 pode ver-se a representação em código C do lugar apresentado na figura 3.2 em PNML.

3.3. TRANSIÇÃO

Uma transição em RdP é representada em PNML como mostra a figura 3.4.

Como se pode observar, retirando novamente as características gráficas que apenas são necessárias para a representação gráfica da rede, encontra-se o “id” e o nome da transição, a prioridade que define quais as transições que disparam primeiro em caso de conflito, os “SignalInputGuards” que definem condições dependentes dos sinais para o disparo da transição, os eventos de entrada que têm de ocorrer para a transição disparar e os eventos de saída que são criados pelo disparo da transição.

```
<transition id="18063163">
  <name>
    <text>B3</text>
    <graphics>
      <offset page="1" x="78" y="21"/>
    </graphics>
  </name>
  <comment>
    <text></text>
    <graphics>
      <offset page="1" x="64" y="30"/>
    </graphics>
  </comment>
  <priority>1</priority>
  <signalInputGuards>
    <signalinputguard>
      <concreteSyntax language="C">
        <text>(insignal180511805 == 1)</text>
      </concreteSyntax>
    </signalinputguard>
  </signalInputGuards>
  <inputEvents>
    <event idRef="inEvent_GATEc3"/>
  </inputEvents>
  <outputEvents>
    <event idRef="outevent1551"/>
  </outputEvents>
  <graphics>
    <position page="1" x="540" y="260"/>
  </graphics>
</transition>
```

Figura 3.4 - Representação de uma transição em PNML.

Cada transição da RdP corresponde em C a uma condição *"if"* dependente das condições necessárias para a transição disparar. Condições tais como: se os lugares de entrada têm o número de marcas que o peso do arco especifica para disparar a transição, ou se os eventos de entrada ocorreram, ou se as condições dos sinais são as necessárias, ou ainda se os "SignalInputGurds" são satisfeitos.

Em seguida, nas linhas a ser executadas caso a condição seja satisfeita são adicionadas as alterações à rede resultantes do disparo da transição como destruição e criação de marcas, ou criação de eventos.

Desta forma, a transição em PNML representada na figura 3.4 teria uma representação em C semelhante à apresentada na figura 3.5.

```
// Transition 18063163 ( B3 )
if(aux_p_18063099_Mark >= 1 && (insignal180511805 == 1) && inEvent_GATEc3 == 1)
 {
    aux_p_18063099_Mark = aux_p_18063099_Mark - 1;
    new_p_18063113_Mark = new_p_18063113_Mark + 1;
    outevent1551 = 1;
}
```

Figura 3.5 - Representação de uma transição em ANSI C.

3.4. ARCO

O arco é representado em PNML da forma apresentada na figura 3.6.

```
<arc id="18063293" source="18063099" target="18063163">
  <type>normal</type>
  <graphics>
    <position page="1" x="462" y="293"/>
  </graphics>
  <inscription>
    <graphics>
      <offset page="1" x="12" y="33"/>
    </graphics>
    <value>1</value>
  </inscription>
</arc>
```

Figura 3.6 - Representação de um arco em PNML.

O arco é definido e usado na fase de geração de código mas não tem uma definição concreta no código gerado em C, sendo usado quando se trata uma transição para definir os lugares de entrada e de saída e as marcas destruídas e criadas, respectivamente, nesses mesmos lugares aquando do disparo da transição. Para cada transição é feita uma lista de arcos de entrada, com todos os arcos que têm a transição como “target” na descrição em PNML e outra de arcos de saída com todos os arcos que têm a transição como “source”. Com estas duas listas, analisando o “source” dos arcos da lista de entrada temos todos os lugares de entrada da transição tal como analisando os “target” dos arcos da lista de arcos de saída temos todos os lugares de saída para a transição. Aquando da análise da transição é usada a “inscription” que representa o peso do arco para definir o número de marcas a serem criadas ou destruídas no lugar ligado.

3.5. ARCO DE TESTE

O arco de teste é definido da mesma forma que os arcos normais, mas a condição que destrói as marcas na variável do lugar não é adicionada para este tipo de arcos.

3.6. SINAL DE ENTRADA

Para a representação em ANSI C de um sinal de entrada, que é representado em PNML como mostra a figura 3.7, é importante ter dois valores. O valor actual do sinal e o valor anterior do sinal, Isto porque será necessário avaliar a variação do sinal para verificar os eventos que ocorrem em cada ciclo de análise.

```
<signal id="inSignal_GATEc3" type="boolean">
  <graphics>
    <position page="1" x="83" y="780"/>
  </graphics>
</signal>
```

Figura 3.7 - Representação de um sinal de entrada em PNML.

Desta forma, os sinais de entrada são representados em C através de duas variáveis. O tipo desta variável depende do valor máximo que o sinal pode atingir sendo *"char"* no caso do tipo do sinal ser "boolean" ou *"int"* no caso de o tipo ser "range". Uma das variáveis guarda o valor recebido do exterior e a outra o valor anterior do sinal

guardado na última iteração. No inicio de cada iteração o valor da variável do valor actual é guardado na variável do valor anterior.

Desta forma a representação do sinal de entrada anterior tem como representação em ANSI C o apresentado na figura 3.8.

```
char inSignal_GATEc3;char last_inSignal_GATEc3;
```

Figura 3.8 - Representação de um sinal de entrada em ANSI C.

3.7. SINAL DE SAÍDA

Um sinal de saída é representado em PNML como mostra a figura 3.9.

```
<signal id="outSignal_Car3MovesBack" type="boolean" value="0">
  <graphics>
    <position page="1" x="723" y="880"/>
  </graphics>
</signal>
```

Figura 3.9 - Representação de um sinal de saída em PNML.

Tal como nos sinais de entrada os sinais de saída são representados em C através de duas variáveis O tipo desta variável depende do valor máximo que o sinal pode atingir sendo *"char"* no caso do tipo do sinal ser "boolean" ou *"int"* no caso de o tipo ser "range". Uma das variáveis guarda o valor recebido do exterior e a outra o valor anterior do sinal guardado da última iteração. Neste caso é no fim de cada iteração que o valor da variável do valor actual é guardado na variável do valor anterior.

Desta forma a representação do sinal de entrada anterior tem como representação em ANSI C o apresentado na figura 3.10.

```
char outSignal_Car3MovesBack;char last_outSignal_Car3MovesBack;
```

Figura 3.10 - Representação de um sinal de saída em ANSI C.

3.8. EVENTO DE ENTRADA

Um evento de entrada (InputEvent) é representado em PNML como mostra a figura 3.11.

```
<event edge="up" id="inEvent_GATEc3" level="0" signal="inSignal_GATEc3">
  <graphics>
    <position page="1" x="383" y="780"/>
  </graphics>
</event>
```

Figura 3.11 - Representação de um evento de entrada em PNML.

Um evento de entrada é representado em C através de uma variável do tipo *"char"* pois o seu valor apenas pode ser zero ou um, ou seja, se ocorreu ou não, reiniciada no fim de cada iteração.

Os eventos de entrada podem ser externos caso o seu valor venha do exterior ou internos caso o seu valor seja dado pela análise do sinal associado. Como o PNML não faz a distinção entre interno e externo essa informação é dada pelo utilizador. Para

todos os eventos externos é retirada a análise do sinal associado que permite a detecção do evento e o valor é recebido do exterior. Neste caso é também dada a possibilidade ao utilizador de apagar o sinal associado.

O Evento apresentado anteriormente em PNML tem como representação em ANSI C a apresentada na figura 3.12.

```
char inEvent_GATEc3;
```

Figura 3.12 - Representação de um evento de entrada em ANSI C.

3.9. EVENTO DE SAÍDA

Tal como o evento de entrada, representado em PNML na figura 3.13, o evento de saída é representado em C através de uma variável do tipo *"char", mas* reiniciada no inicio de cada iteração.

Os eventos de saída, tal como os de entrada podem ser externos caso o seu valor seja enviado directamente para o exterior ou internos caso o seu valor seja usado na análise do sinal associado. Tal como para os eventos internos é dada a possibilidade ao utilizador de apagar o sinal associado.

```
<event edge="up" id="outEvent_GATEc3" level="0" signal="outSignal_GATEc3">
  <graphics>
    <position page="1" x="1063" y="80"/>
  </graphics>
</event>
```

Figura 3.13 - Representação de um evento de saída em PNML.

A representação em ANSI C do evento de saída apresentado em cima em PNML está apresentada na figura 3.14.

```
char outevent1569;
```

Figura 3.14 - Representação de um evento de saída em ANSI C.

4. DESENVOLVIMENTO DA FERRAMENTA

Depois da definição das regras de tradução dos elementos das IOPT para código C partiu-se para o desenvolvimento de uma ferramenta para geração automática deste mesmo código C. Neste capitulo será apresentada a ferramenta desenvolvida para a tradução das redes de Petri IOPT existentes no inicio da execução deste trabalho, as usadas no desenvolvimento do projecto FORDESIGN [Fordesign, 07]. Durante a desenvolvimento desta ferramenta existiram evoluções nas redes IOPT que resultaram na definição de uma nova gramática que acrescenta algumas alterações às mesmas. Foi decidido terminar o desenvolvimento da ferramenta com a gramática antiga em RelaxNG e em seguida partir para o desenvolvimento de uma nova ferramenta para trabalhar com a nova gramática produzido em Ecore. Esta será apresentada mais à frente tal como as alterações e as regras de tradução necessárias para traduzir as definições e elementos/anotações acrescentados às IOPT. Para este efeito foi usada a ferramenta de desenvolvimento Visual Studio .Net 2008.

A ferramenta, que pode ser usada por um utilizador humano ou por outra ferramenta, disponibiliza como casos de uso, carregar um ficheiro PNML (pode ser editado manualmente ou por uma ferramenta como o SnoopyIOPT), gerar um conjunto de ficheiros C, definir eventos de entrada e de saída e salvar o conjunto de ficheiros C gerados. Estes casos de uso estão representados na figura 4.1.

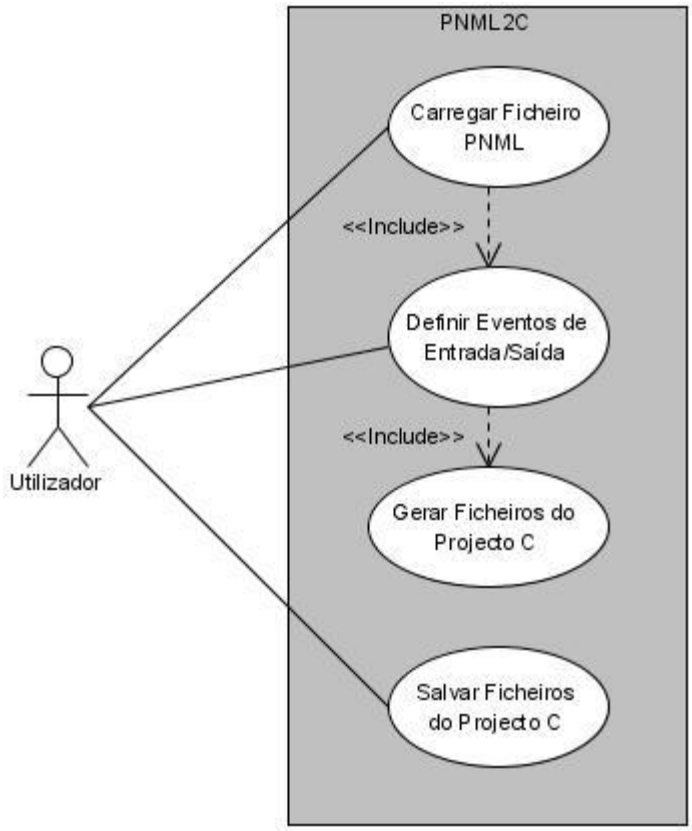

Figura 4.1 - Diagrama de casos de uso.

4.1. INTERFACE

Foram desenvolvidos dois interfaces diferentes para a ferramenta. Um para ser usado por utilizadores humanos, com ambiente gráfico e outro para acesso por instruções em linha de comandos para integração com outras ferramentas.

4.1.1. INTERFACE GRÁFICO

O interface gráfico é feito através de uma *form* e permite ao utilizador comunicar com a ferramenta através de botões, como o apresentado na figura 4.2, permitindo, neste caso, chamar a função de carregamento onde o utilizador vai escolher o ficheiro PNML de entrada na ferramenta.

Outro ponto importante de interacção do ambiente gráfico com o utilizador é a selecção dos eventos internos e externos que é feito através de uma janela onde são listados os vários eventos e sinais e é disponibilizada um *checkbox* para seleccionar caso o evento seja interno. Um exemplo deste interface está representado na figura 4.2.

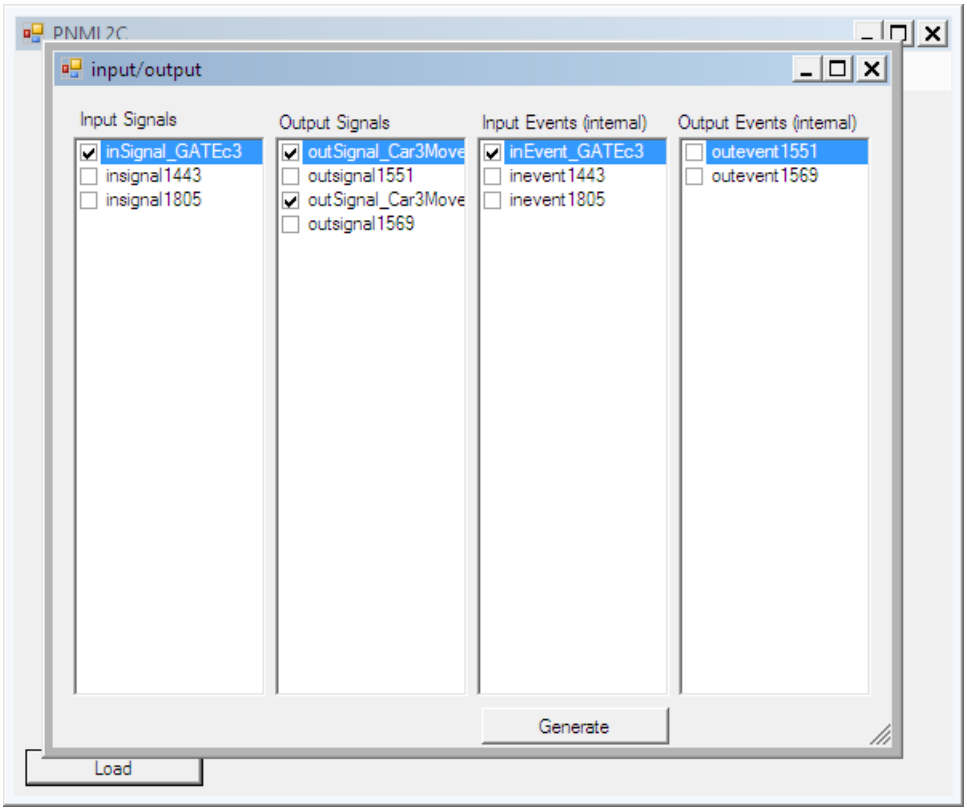

Figura 4.2 - Selecção de eventos internos e externos.

A explicação do funcionamento detalhado deste interface gráfico encontra-se no manual de utilização da ferramenta, em anexo nesta dissertação.

4.1.2. INTERFACE EM LINHA DE COMANDOS

No interface de linha de comandos é usada uma “string” para comunicar com a ferramenta. Nesta “string” é enviada toda a informação necessária para a ferramenta fazer a geração do código.

“PNML2C ficheiro PNML Directoria saída Opções”

A “string” começa com o nome da aplicação e em seguida recebe a informação necessária.

Depois do nome é adicionado o ficheiro PNML a tratar com a sua localização, em seguida a pasta onde será criada a pasta com os ficheiros e em seguida a informação necessária do exterior sobre os eventos e os sinais já falada anteriormente. Esta informação é separada por um espaço.

A informação enviada começa sempre com um identificador. Para cada tipo de informação existe um identificador e em seguida um conjunto de dados dentro de parênteses separados por uma vírgula. Na tabela 4.1 seguinte estão representados os identificadores para cada tipo de informação.

Acção a Realizar Pela Ferramenta	Identificador
Remover Sinal de Saída	-sor
Remover Sinal de Entrada	-sir
Incluir Evento Saída no interface com o exterior	-evo
Incluir Evento Entrada como vindo do exterior	-evi

Tabela 4.4.1- Identificadores das acções a realizar pela ferramenta.

Vai-se agora usar o mesmo exemplo apresentado na secção anterior, usando o interface gráfico, aplicado o interface de linha de comandos.

Neste caso ter-se-á de introduzir, como comando, a seguinte “string”:

PNML2C c:\car3.pnml c:\code –sir(insignal1443,insignal1805)

–sor(outsignal1551,outsignal1569) –evi(inevent1443,inevent1805)

–evo(outevent1551,outevent1569)

Traduzindo esta “string” temos:

O texto “c:\car2.pnml”, representa o endereço do ficheiro PNML a abrir e ler.

Em seguida temos “-sir(insignal1443,insignal1805)”. Esta parte define os sinais de entrada a ser removidos, depois do identificador aparecem entre parênteses os sinais separados por uma vírgula.

Depois encontra-se o texto “-sor(outsignal1551,outsignal1569)” que define os sinais de saída a ser removidos, depois do identificador aparecem entre parênteses os sinais separados por uma vírgula.

No seguimento vem “-evi(inevent1443,inevent1805)” que define os eventos de entrada a definir como vindos do exterior, depois do identificador aparecem entre parênteses os eventos separados por uma vírgula.

Por fim temos “-evo(outevent1551,outevent1569)”. Esta parte representa os eventos de saída a incluir no interface com o exterior, depois do identificador aparecem entre parênteses os eventos separados por uma vírgula.

Todos os pedaços da “string” apresentados isoladamente anteriormente são separados por um espaço.

4.2. CÓDIGO GERADO

A ferramenta desenvolvida recebe um ficheiro em formato PNML com as especificações da rede IOPT a traduzir e cria um conjunto de 5 ficheiros preparados para executar a rede exactamente como definida na Rede IOPT. Os ficheiros gerados são três ficheiros “.C” (main.c, ioptc.c e functions.c) e os respectivos Header (.H) dos ficheiros que necessitam (ioptc.h e functions.h). O ficheiro “*functions”* tem definidas funções que são usadas pelo ficheiro *“ioptc”.* Neste está definida a inicialização da rede e a execução desta. No ficheiro “*main*” é utilizado o ficheiro *“ioptc”* estando preparado para receber o interface para a plataforma onde será executado. O interface pode ser definido por um humano ou por uma ferramenta de definição de interface. Este conjunto de ficheiros gerados está representado na figura 4.3.

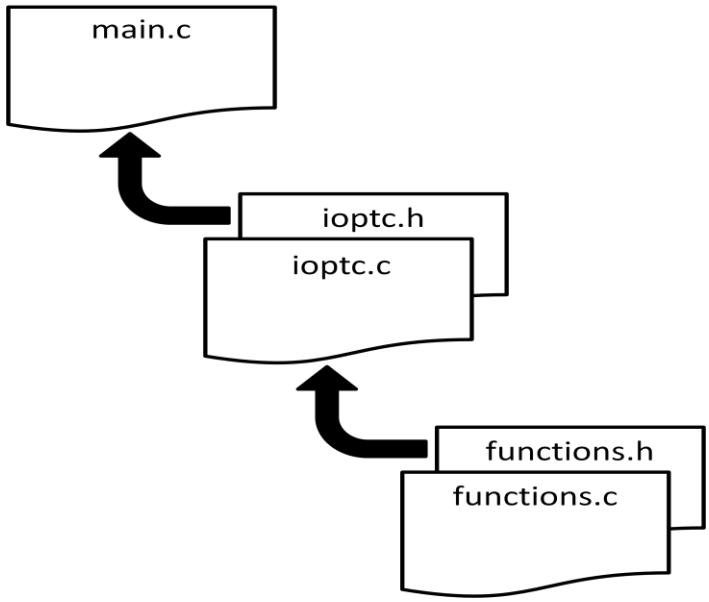

Figura 4.3 - Representação da estrutura do projecto criado.

4.2.1. FICHEIROS "FUNCTIONS" (.C E .H)

Os ficheiros "functions" gerados são os ficheiros do projecto onde são guardadas as funções a ser usadas no tratamento das variáveis na execução da rede. Estes ficheiros foram criados principalmente para tornar o código final mais legível e fácil de entender.

No ficheiro "functions.h" são declaradas todas as variáveis do projecto, todos os sinais de entrada (valores actual e anterior), todos os sinais de entrada (valores actual e anterior), todos os eventos de entrada e de saída e para cada lugar as três variáveis que o definem. Para além dessas variáveis são declaradas as funções que serão definidas no ficheiro "functions.c".

No ficheiro "functions.c" são definidas 2 funções. Estas funções são "Analyse_Input_Events" e "Analyse_Output_Events" apresentadas em seguida:

Analyse_Input_Events.

A função “Analyse_Input_Events” analisa os sinais de entrada de forma a verificar se a sua variação implica a geração de algum evento dependente da variação do sinal. Para isso, para todos os eventos é analisado se o valor actual e o valor anterior do sinal são os necessários para satisfazer a condição para a existência do evento. Por exemplo o evento “inEvent_GATEc3” representado em PNML na figura 3.11 está dependente do sinal “inSignal_GATEc3” e ocorre quando existir um flanco ascendente a partir de zero neste sinal. Pode entender-se então que este evento ocorre quando o valor actual do sinal é um e o valor anterior era zero.

Desta forma para criar o evento é acrescentada à função uma condição como a apresentada na figura 4.4.

```
void Analyse_Input_Events(void)
{
    if(inSignal_GATEc3 == 1 && last_inSignal_GATEc3 == 0)
        inEvent_GATEc3 = 1;
}
```

Figura 4.4 - Exemplo da função "Analyse_Input_Events".

Como se pode verificar o evento será um sempre que o sinal tenha sido zero na iteração anterior e um na iteração actual. Caso contrário manter-se-á em zero, valor com o qual é iniciado no inicio de cada iteração.

Analyse_Output_Events

De forma semelhante, a função “Analyse_Output_Events” analisa os eventos de saída gerados pela execução da rede para alterar o valor do sinal de saída associado. Para esse efeito é analisado se o valor do evento e o valor do sinal na última iteração são os necessários para alterar o valor do sinal.

```
void Analyse_Output_Events(void)
{
    if(last_outSignal_GATEc3 == 0 && outEvent_GATEc3 == 1)
        outSignal_GATEc3 = 1;
}
```

Figura 4.5 - Exemplo da função "Analyse_Output_Events".

A figura 4.5 mostra o exemplo de uma função “Analyse_Output_Events” que trata o Evento de saída “outEvent_Gatec3” que quando ocorre afecta o sinal “outSignal_GATEc3” passando-o de zero(0) para um(1)

4.2.2. FICHEIROS IOPTC (.C E .H)

É nestes ficheiros que realmente se define a execução da rede, o ficheiro ioptc.h começa por ser incluído o ficheiro functions.h. Em seguida são definidas todas as variáveis do projecto, todos os sinais de entrada (valores actual e anterior), todos os sinais de entrada (valores actual e anterior), todos os eventos de entrada e de saída e

para cada lugar as três variáveis que o definem. Finalmente são declaradas as duas funções que serão definidas no ficheiro ioptc.h, start() que inicializa a rede e run() que executa uma iteração na rede.

No ficheiro ioptc.c temos a definição das funções referidas anteriormente.

start()

A função start() tem como objectivo, quando é chamada, inicializar as variáveis do projecto com os valores iniciais da rede.

As variáveis do valor actual dos sinais de entrada e de saída e dos eventos de entrada e de saída são inicializadas a zero e as variáveis com os valores actuais das marcações dos lugares são inicializadas com os valores das marcações iniciais dos lugares que estão definidos no PNML.

A figura 4.6 mostra o exemplo de uma função Start() para uma RdP com um sinal de entrada (inSignal_Gatec3), dois sinais de saída (outSignal_Car3MovesBack e outSignal_Car3Moves), três eventos de entrada (inEvent_Gatec3, inevent1443, inevent1805), dois eventos de saída (outevent1569, outevent1551) e quatro lugares (Car3_move_back, Car3_ready, Car3_move e Car3_at_end).

```
char start(void)
{
//----------------Init Input Signals Values---------------------
      inSignal_GATEc3 = 0;
//--------------------------------------------------------------

//----------------Init Output Signals Values---------------------
      outSignal_Car3MovesBack = 0;
      outSignal_Car3Moves = 0;
//--------------------------------------------------------------

//----------------Init Input Events Values---------------------
      inEvent_GATEc3 = 0;
      inevent1443 = 0;
      inevent1805 = 0;
//--------------------------------------------------------------

//----------------Init Output Events Values---------------------
      outevent1569 = 0;
      outevent1551 = 0;
//--------------------------------------------------------------

//----------------Init Place Marks---------------------
      p_18061303_Mark = 0; // Place Car3_move_back
      p_18061317_Mark = 1; // Place Car3_ready
      p_18063099_Mark = 0; // Place Car3_move
      p_18063113_Mark = 0; // Place Car3_at_end
//--------------------------------------------------------------

      return 1;
}
```

Figura 4.6 - Exemplo da função Start().

run()

Na função run() é executada uma iteração da rede verificando quais as transições que estão habilitadas a disparar e executando o seu disparo colocando a rede no estado seguinte.

Para obedecer ao fluxo já apresentado na figura 3.1 começa-se por inicializar a zero para todos os lugares a variável onde serão guardadas as novas marcas como exemplificado na figura 4.7

```
new_p_18061303_Mark = 0; // Place Car3_move_back
new_p_18061317_Mark = 0; // Place Car3_ready
new_p_18063099_Mark = 0; // Place Car3_move
new_p_18063113_Mark = 0; // Place Car3_at_end
```

Figura 4.7 - Inicialização das variáveis onde são guardadas as novas marcas.

Em seguida, como mostra a figura 4.8, para cada lugar é copiado o valor da marcação actual para a variável auxiliar onde serão consumidas as marcas durante a execução da iteração.

```
aux_p_18061303_Mark = p_18061303_Mark; // Place Car3_move_back
aux_p_18061317_Mark = p_18061317_Mark; // Place Car3_ready
aux_p_18063099_Mark = p_18063099_Mark; // Place Car3_move
aux_p_18063113_Mark = p_18063113_Mark; // Place Car3_at_end
```

Figura 4.8 - Cópia da Marcação dos lugares para uma variável auxiliar.

No passo seguinte, como apresentado na figura 4.9, são copiados os valores dos sinais de saída para as variáveis do valor anterior de forma a guardar este valor para a análise das alterações nos sinais.

```
//Save Output Signals
last_outSignal_Car3MovesBack = outSignal_Car3MovesBack;
last_outSignal_Car3Moves = outSignal_Car3Moves;
```

Figura 4.9 - Cópia dos valores dos sinais de saída para a variável do último valor.

Depois são inicializados a zero os eventos de saída e é feita a análise dos eventos dos sinais de entrada para verificar se ocorreu algum evento de saída executando a já referida função “Analyse_Input_Events”, como mestra a figura 4.10

```
// Reset Output Events
outevent1569 = 0;
outevent1551 = 0;

Analyse_Input_Events();
```

Figura 4.10 - Inicialização dos sinais de saída e análise da ocorrência de eventos de entrada.

Neste momento começa a análise das transições. De forma a resolver conflitos as transições têm prioridades e aproveitando o facto de o código em C ser sequencial este problema é resolvido ordenando as transições por prioridade das mais prioritárias para as menos. Tendo em conta que a prioridade um é a mais prioritária e quanto maior for o valor menos prioritária é a transição as transições são ordenadas de forma crescente deste valor.

Em seguida começa-se o tratamento de cada transição. Para cada uma é adicionada uma condição “if()”. Dentro deste “if()” temos uma condição com um conjunto de equações divididas por uma condição “e” (&&).

Começa-se por adicionar as condições que definem se os lugares de entrada têm o numero de marcas necessárias para habilitar a transição para isso é adicionada à condição uma equação que verifica se na marcação auxiliar de cada lugar de entrada existe um numero de marcas maior ou igual (>=) ao valor do peso do arco que o liga à transição em análise.

Em seguida são adicionados os “SignalInputGuards”, que são condições associadas a sinais de entrada que condicionam o disparo da transição. Estas estão definidas no PNML considerando uma linguagem especifica e são adicionadas todas as que têm linguagem C exactamente como são apresentadas.

Finalmente são adicionadas as equações que verificam se as variáveis dos eventos de entrada associados à transição têm o valor um, ou seja, se ocorreram.

A construção destas condições está descrita nos passos seguintes:

1. *reset Lista de Lugares de Entrada*
2. *reset Lista de Arcos de Entrada*
3. *reset Lista de Lugares de Saída*
4. *reset Lista de Arcos de Saída*
5. *adiccionar Texto “if(“*
6. *para todos os lugares de Entrada faz*
7. *se arco de teste então*
8. *adiccionar Texto “p_source place id_Mark == Source Arc inscription”*
9. *senão*

10. adicionar texto “aux_p_source place id_Mark == Source Arc inscription”
11. fim se
12. fim para
13. para todos Signal Input Guards faz
14. se a Linguagem é C então
15. adiccionar texto “&& Signal input gard text”
16. fim se
17. Fim para
18. Para Todos os Eventos de entrada faz
19. adicionar texto “&& InputEvent == 1”
20. fim para
21. adiccionar texto “)”
22. (adiccionar acções associadas ao disparo)

Dentro de chavetas a seguir ao “if()” são adicionadas as alterações à rede que o disparo da transição efectua. Estas são tratadas como se mostra nas linhas seguintes:

1. para todos os lugares de saída faz
2. adiccionar texto “aux_p_target place id_Mark -= Target Arc inscription”
3. fim para
4. para todos os lugares de entrada faz
5. se não for arco de teste então
6. adiccionar texto “aux_p_source place id_Mark += Source Arc inscription”
7. fim se
8. fim para
9. para todos os eventos de saída da transição faz
10. para todos os eventos de saída faz
11. se o evento de saída da transição é evento de saída então
12. se edge for up então
13. adiccionar texto “aux_Signal += Event Level”
14. fim se
15. Senão se edge for down então
16. adicionar texto “aux_Signal -= Event Level”
17. fim se

18. fim se

19. fim para

20. fim para

Começa-se por subtrair, para cada lugar de entrada, o valor do peso do arco que o liga à transição na variável auxiliar que representa o lugar. É analisado sempre inicialmente se o arco é de teste. No caso de ser um arco de teste não é adicionada a condição de subtracção de marcas.

Em seguida são adicionadas na variável de novas marcas dos lugares de saída o número de marcas igual ao valor do peso do arco que liga o lugar à transição.

Finalmente são colocadas as variáveis que representam os Eventos de saída criados pelo disparo da transição com o valor um. Na figura 4.11 é apresentado o exemplo de duas transições analisadas.

```
// Transition 18061331 ( GO_m3 )
if(aux_p_18061317_Mark >= 1 && inevent1443 == 1)
 {
    aux_p_18061317_Mark = aux_p_18061317_Mark - 1;
    new_p_18063099_Mark = new_p_18063099_Mark + 1;
}

// Transition 18061349 ( A3 )
if(aux_p_18061303_Mark >= 1 && inEvent_GATEc3 == 1)
 {
    aux_p_18061303_Mark = aux_p_18061303_Mark - 1;
    new_p_18061317_Mark = new_p_18061317_Mark + 1;
    outevent1569 = 1;
}
```

Figura 4.11 - Análise das transições em ANSI C.

Depois de tratadas todas as transições é feito o tratamento da marcação resultante, adicionando para cada lugar as marcas que restaram na variável auxiliar às marcas criadas na variável de novas marcas e o valor resultante é colocado na variável do valor actual de marcas, como mostra a figura 4.12.

```
// Actualize Marks
p_18061303_Mark = aux_p_18061303_Mark + new_p_18061303_Mark;
p_18061317_Mark = aux_p_18061317_Mark + new_p_18061317_Mark;
p_18063099_Mark = aux_p_18063099_Mark + new_p_18063099_Mark;
p_18063113_Mark = aux_p_18063113_Mark + new_p_18063113_Mark;
```

Figura 4.12 - Calculo do novo valor de marcações dos lugares.

Depois são copiados os valores actuais dos sinais de entrada para a variável do valor anterior, como apresentado na figura 4.13.

```
//Save Input Signals
last_inSignal_GATEc3 = inSignal_GATEc3;
```

Figura 4.13 - Cópia dos valores dos sinais de entrada para a variável do último valor.

Logo em seguida, como mostra a figura 4.14, são iniciados todos os eventos de entrada a zero e os sinais de saída com o seu valor inicial.

```
// Reset Input Events
inEvent_GATEc3 = 0;
inevent1443 = 0;
inevent1805 = 0;

outSignal_Car3MovesBack = 0;
outSignal_Car3Moves = 0;
```

Figura 4.14 - Inicialização dos eventos de entrada e dos sinais de saída.

Neste momento são analisadas as “SignalOutputActios” associadas a todos os lugares. As “SignalOutputActios” são condições definidas por linguagem e que são executadas quando o lugar a que estão associadas está marcado. Desta forma é analisada cada uma delas colocando para cada uma, uma condição “if()” que executa a condição apresentada no PNML em C, caso o lugar associado esteja marcado e o valor do sinal em questão seja inferior ao valor para o qual a condição o altera.

Finalmente é chamada a função “Analyse_Output_Events” para alterar os sinais que os eventos internos criados na iteração afectam. Estes dois últimos passos são apresentados na figura 4.15.

```
if(p_18061303_Mark >= 1)
  {
    if(outSignal_Car3MovesBack < 1);
      outSignal_Car3MovesBack = 1;
}
if(p_18063099_Mark >= 1)
  {
    if(outSignal_Car3Moves < 1);
      outSignal_Car3Moves = 1;
}

Analyse_Output_Events();
```

Figura 4.15 - Análise das “SignalOutputActios” e dos eventos de saída.

4.2.3. FICHEIRO MAIN.C

No ficheiro main.c é executada a rede. Este está preparado para receber o interface com o exterior e também qualquer cálculo adicional necessário para as variáveis recebidas do exterior.

Começa por ser incluído o ficheiro “iopt.h” de forma a permitir a utilização das funções deste. Em seguida são declaradas todas as variáveis do projecto. Esta será a declaração principal, pois em todos os outros ficheiros são declaradas como "extern" ficando associadas a estas.

Em seguida tem-se a função “main()”, apresentada na figura 4.16, que executará todo o projecto.

```
int main()
{
    start();

    //Insert your code here...

    while(/*Set while condition here...*/)
    {
        //Set Input Signals
        //Set Input Events

        run();

        //Put Output Signals
        //Put Output Events
    }

    //Insert your code here...

    return 0;
}
```

Figura 4.16 - Função "main" como é gerada.

Começa-se por chamar a função “start()” para inicializar a rede. E em seguida é colocado um ciclo “while” com a função run() no seu interior. Todo o código adicional a este ficheiro deve ser criado pelo programador do interface.

5. META-MODELO IOPT

Com o desenvolvimento da nova gramática das IOPT e a criação do meta-modelo (ECORE) foram efectuadas alterações às RdP IOPT. Para a tradução destas foi criada uma nova ferramenta que as regras já definidas anteriormente acrescentando algumas alterações para permitir traduzir as alterações feitas às RdP IOPT.

Por motivos que se prendem com a interacção das várias ferramentas em desenvolvimento para a nova gramática foi decidido desenvolver a nova ferramenta em linguagem “mofscript” em plataforma eclipse. Como em grande parte o código gerado é semelhante e as regras de tradução são as mesmas mudando apenas a linguagem em que se implementa, nesta secção vai-se apenas apresentar as alterações à gramática das IOPT e a forma usada para as traduzir em ANSI C.

5.1. EVENTOS AUTÓNOMOS

Uma das alterações feitas às IOPT foi o adicionar de um novo tipo de eventos, os eventos autónomos que não estão dependentes nem afectam nenhum sinal e o seu valor é recebido ou enviado para o exterior, dependendo se é um evento de entrada ou de saída. Com estes eventos não é necessário adicionar nada às regras de tradução para ANSI C. Esta novidade permite aliás que deixe de ser necessário ser o utilizador a dizer quais os eventos que são externos e não estão associados a sinais.

5.2. SINAIS COM MÚLTIPLAS AFECTAÇÕES

Na gramática antiga das IOPT a forma de definir o valor a dar a um sinal que numa mesma iteração é afectado com valores diferentes era omissa. Definiu-se como forma de resolver o problema, escolher entre todos os valores o mais afastado do valor por omissão.

Nesta nova gramática esse problema já se encontra tratado sendo adicionado aos sinais essa informação. O valor pode ser dado pelo valor mais próximo da omissão, o mais longe da omissão, o máximo, o mínimo, a média ou a mediana entre os vários valores a serem dados ao sinal.

Esta alteração é a que provoca mais alterações ao código gerado. Para além de cinco funções adicionadas ao ficheiro “functions”, para ordenar um array, obter o valor do máximo, mínimo, média e mediana, que serão apresentadas mais à frente, a representação do sinal em ANSI C apenas com duas variáveis do tipo “char” ou “long” passa a não ser suficiente para o seu tratamento.

O sinal passa a ser declarado, para além das duas variáveis já apresentadas com o valor actual e o valor anterior, por uma variável onde é guardado o número de vezes que o sinal foi afectado nessa iteração e um array onde vão ser guardados esses valores. O Sinal em ANSI C passa então a ter o aspecto apresentado na figura 5.1.

```
//----------------Output Signals Values---------------------
char outSignal_Car3MovesBack;char last_outSignal_Car3MovesBack;
char outSignal_Car3MovesBack_values[4];
char outSignal_Car3MovesBack_count;
```

Figura 5.1- Declaração de um Sinal de saída.

Cada vez que é dado um novo valor ao sinal durante a iteração é adicionado um valor ao array e incrementado o contador. Desta forma em cada local onde é afectado o sinal é adicionado o código apresentado na figura 5.2.

```
outSignal_Car3MovesBack_values[outSignal_Car3MovesBack_count] = 1;
outSignal_Car3MovesBack_count++;
```

Figura 5.2 - Alteração do sinal de saída.

A análise do valor a dar ao sinal é feito neste caso no final da iteração, em seguida à análise das marcações dos lugares para determinar valores do sinal e antes da análise dos eventos. Para cada sinal de saída é analisado o valor segundo a forma que é dada para o sinal.

O cálculo do valor resultante do sinal é realizado no final de cada iteração, como se ilustra na figura 5.3.

```
outSignal_Car3MovesBack = max(outSignal_Car3MovesBack_values,outSignal_Car3MovesBack_count);
outSignal_Car3Moves = median(outSignal_Car3Moves_values,outSignal_Car3Moves_count);
```

Figura 5.3 - Determinação dos valores dos sinais de saída.

Este conjunto de código é colocado entre a análise das alterações aos sinais através das marcações dos lugares e a análise dos eventos de saída.

5.3. EXPRESSÕES

Como já tinha sido referido anteriormente nesta dissertação, as expressões usadas para tratar guardas nas transições e alterações do valor dos sinais com a análise das marcações dos lugares, eram dadas directamente no código PNML com a expressão definida para cada linguagem.

O novo meta-modelo introduz como novidade a expressão em RdP IOPT.

Esta expressão está definida no PNML de uma forma totalmente independente da linguagem.

São disponibilizados os operadores lógicos *AND e OR* e os operadores de comparação *EQUALS, NOT_EQUALS, GREATER_THAN, LESSER_THAN, GREATHER_OR_EQUAL_THAN e LESSER_OR_EQUAL_THAN*.

Esta introdução permite ao modelador de sistemas em RdP IOPT uma maior abstracção quanto à linguagem ou linguagens em que o sistema vai ser implementado, tornando também a descrição do sistema mais independente da plataforma a implementar.

Para a ferramenta apresentada neste capítulo, esta introdução não traz qualquer tipo de alterações ao nível do código gerado. Apenas ao nível da geração de código há alterações.

De forma a traduzir uma expressão em RdP IOPT numa expressão em ANSI C criou-se uma função que à medida que vai analisando a expressão IOPT vai criando uma expressão em ANSI C que será devolvida pela função em formato de “string”.

De forma a entender a forma como funciona esta função vejamos a seguinte condição definida em RdP IOPT.

“insignal1 EQUALS 0 AND insignal2 GREATER_THAN 1”

Esta expressão teria como tradução para ANSI C a condição seguinte.

“insignal1 == 0 && insignal2 >=1”.

Esta expressão seria agora tratada da mesma forma que a que vinha anteriormente do PNML definida já para cada linguagem.

6. EXEMPLOS DE APLICAÇÃO

Para validar a ferramenta começou-se por usar alguns exemplos simples de aplicação.

Neste momento existem algumas teses de mestrado a usar a aplicação com o intuito de a validar de forma mais eficaz.

Um dos exemplos usados para validar a aplicação introduzido em [Silva, 85] vai ser apresentado nesta secção de forma a facilitar a percepção da forma de tradução da ferramenta.

O objectivo é produzir o controlador para um sistema de automação composto por três carros para transporte de mercadorias, como mostra a figura 6.1. Os carros movem-se para frente e para trás entre os dois pontos finais (A e B). Cada carro tem a sua própria trajectória e velocidade, mas devem começar a mover-se todos ao mesmo momento. Começam a avançar quando todos estão em posição de repouso, ou seja, no ponto A, e o botão GO é pressionado. O movimento para trás deve começar quando todos os carros estão na sua posição final, ou seja, no ponto B, e o botão BACK é activado. O controlador tem seis sensores de presença como sinais de entrada (três na posição inicial A e três na posição final B), bem como um botão GO (para iniciar o movimento para a frente) e um botão BACK (para iniciar o movimento para trás). Como saídas, o controlador terá dois tipos de sinais: um deles para controlar os motores dos três carros e outro que indica a direcção do movimento, como mostrado na figura 6.1.

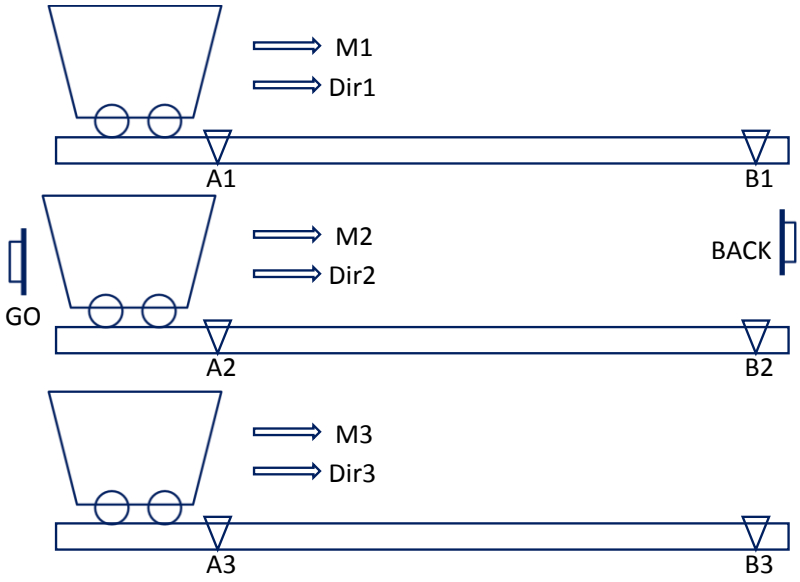

Figura 6.1- Exemplo de aplicação: Sistema de controlo de três carros.

Foram usadas duas abordagens. Uma baseada num controlador centralizado e outra num controlador distribuído onde se partiu a rede em 3 sub-redes.

Os ficheiros criados resultaram nas dimensões, em número de linhas, apresentadas na tabela 6.1.

	Lugares	Transições	Nós	main.c	functionsc.h	functionsc.c	ioptc.h	ioptc.c	Total
3 carros	12	8	20	25	51	19	35	177	307
carro 1	8	8	16	24	47	13	32	147	263
carro 2/3	4	4	8	16	25	9	18	77	145
10 carros	40	22	62	112	173	71	110	565	1031
20 carros	80	42	122	212	333	131	210	1105	1991

Tabela 6.1 - Dimensões dos ficheiros gerados em número de linhas.

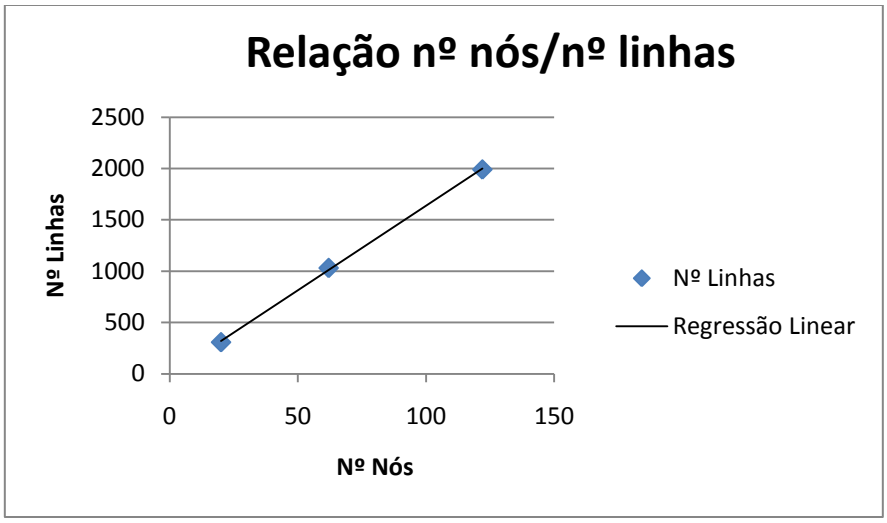

Figura 6.2- Relação entre o nº de nós da RdP e o nº de linhas do código gerado.

Ao serem colocados num microcontrolador PIC18F4520 o projecto criado para cada controlador tem as dimensões apresentadas na figura 6.3.

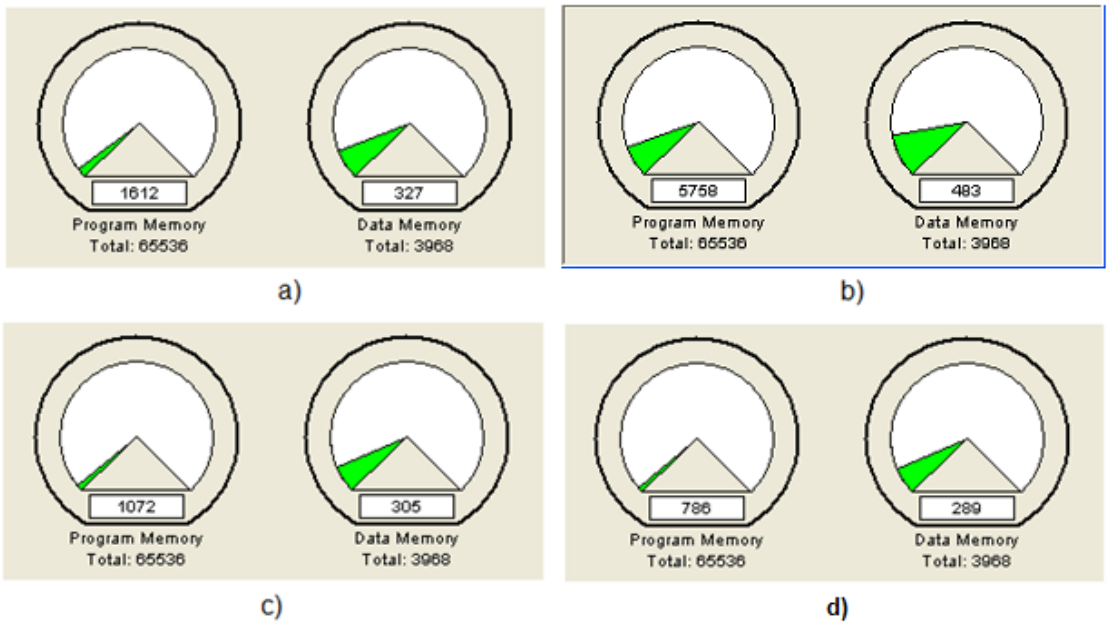

Figura 6.3 - Dimensão do projecto dos controladores.

a) 3 carros (centralizado); b) 10 carros (centralizado);

c) carro 1 (distribuido); d)carro 2 ou 3 (distribuido).

Pode verificar-se pela análise do número de linhas e da dimensão do projecto que o aumento da dimensão do código gerado é aproximadamente proporcional ao aumento do modelo. Nota-se também que o ficheiro que mais é afectado em dimensão com o aumento do modelo é o ficheiro “iopt.c”. Para os exemplos usados o aumento do numero de linhas e da memória de programa é aproximadamente linear em relação ao aumento do numero de carros. Esta linearidade também se verifica na relação entre o numero de nós de uma RdP e o numero de linhas geradas como se pode verificar através da figura 6.2.

6.1. CONTROLADOR CENTRALIZADO

De forma a utilizar redes de Petri como linguagem de especificação do sistema, foi produzido o modelo IOPT apresentado na figura 6.4. Este modelo pode ser usado para gerar automaticamente o código de aplicação para o controlador do sistema.

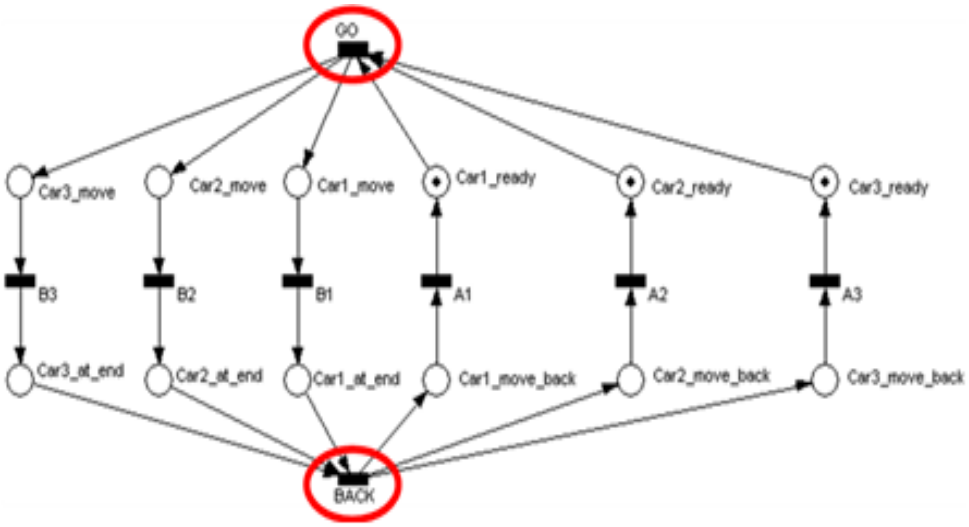

Figura 6.4 - Modelo em RdP IOPT do Exemplo.

Aplicando o PNML gerado para este modelo no gerador de código são obtidos cinco ficheiros já referenciados durante esta dissertação. Todos os ficheiros gerados para este exemplo podem ser encontrados em anexo desta dissertação. Nesta secção apenas vão ser representadas a declaração das variáveis e as funções “start()” e “run()” do ficheiro “ioptc.c”, pois são as que mais representam a execução da RdP.

No código apresentado abaixo, mostra-se a declaração das variáveis a ser usadas no projecto. São declaradas as variáveis que representam os sinais de entrada, sinais de saída, eventos de entrada, eventos de saída e lugares da RdP.

```
//----------------Input Signals Values----------------------
char inSignal_GO;char last_inSignal_GO;
(…)
char inSignal_GATEc3;char last_inSignal_GATEc3;
//----------------------------------------------------------

//----------------Output Signals Values---------------------
char outSignal_Car3Moves;char last_outSignal_Car3Moves;
(…)
char outSignal_Car3MovesBack;char last_outSignal_Car3MovesBack;
//----------------------------------------------------------

//----------------Input Events Values----------------------
char inEvent_GO;
(…)
char inEvent_GATEc3;
//----------------------------------------------------------

//----------------Output Events Values---------------------
char outEvent_Car2Moves;
(…)
char outEvent_Car3MovesBack;
//----------------------------------------------------------      |

//--------------------Place Marks-------------------------
char p_1605_Mark;char aux_p_1605_Mark;char new_p_1605_Mark;
(…)
char p_2112_Mark;char aux_p_2112_Mark;char new_p_2112_Mark;
//----------------------------------------------------------
```

A função “start()” inicializa o sistema com os valores definidos no ficheiro PNML para os sinais de entrada e saída e para as marcações dos lugares e a zero os eventos de entrada e saída.

```
char start(void)
{
//----------------Init Input Signals Values----------------------
    inSignal_GO = 0;
    (…)
    inSignal_GATEc3 = 0;
//---------------------------------------------------------

//----------------Init Output Signals Values----------------------
    outSignal_Car3Moves = 0;
    (…)
    outSignal_Car3MovesBack = 0;
//---------------------------------------------------------

//----------------Init Input Events Values----------------------
    inEvent_GO = 0;
     (…)
    inEvent_GATEc3 = 0;
//---------------------------------------------------------

//----------------Init Output Events Values----------------------
    outEvent_Car2Moves = 0;
    (…)
    outEvent_Car3MovesBack = 0;
//---------------------------------------------------------


//----------------Init Place Marks----------------------
    p_1605_Mark = 0; // Place Car1_move
    (…)
    p_2112_Mark = 0; // Place Car3_at_end
//---------------------------------------------------------

    return 0;
}
```

Finalmente a função “run()” que representa uma iteração do sistema é apresentada no código seguinte.

```
char run(void)
{

    new_p_1605_Mark = 0; // Place Car1_move
    (…)
```

```
        new_p_2112_Mark = 0; // Place Car3_at_end

        aux_p_1605_Mark = p_1605_Mark; // Place Car1_move
        (…)
        aux_p_2112_Mark = p_2112_Mark; // Place Car3_at_end

        //Save Output Signals
        last_outSignal_Car3Moves = outSignal_Car3Moves;
        (…)
        last_outSignal_Car3MovesBack = outSignal_Car3MovesBack;

        // Reset Output Events
        outEvent_Car2Moves = 0;
        (…)
        outEvent_Car3MovesBack = 0;

        Analyse_Input_Events();


        // Transition 1443 ( GO )
        if(aux_p_1647_Mark >= 1 && aux_p_1661_Mark >= 1 && aux_p_1675_Mark >= 1 &&
inEvent_GO == 1)
         {
           aux_p_1647_Mark = aux_p_1647_Mark - 1;
           aux_p_1661_Mark = aux_p_1661_Mark - 1;
           aux_p_1675_Mark = aux_p_1675_Mark - 1;
           new_p_1605_Mark = new_p_1605_Mark + 1;
           new_p_1619_Mark = new_p_1619_Mark + 1;
           new_p_1633_Mark = new_p_1633_Mark + 1;
        }


        (…)

        // Transition 1805 ( BACK )
        if(aux_p_1717_Mark >= 1 && aux_p_1703_Mark >= 1 && aux_p_2112_Mark >= 1 &&
        inEvent_BACK == 1)
         {
           aux_p_1717_Mark = aux_p_1717_Mark - 1;
           aux_p_1703_Mark = aux_p_1703_Mark - 1;
           aux_p_2112_Mark = aux_p_2112_Mark - 1;
           new_p_1731_Mark = new_p_1731_Mark + 1;
           new_p_1745_Mark = new_p_1745_Mark + 1;
           new_p_1759_Mark = new_p_1759_Mark + 1;
        }

        // Actualize Marks
        p_1605_Mark = aux_p_1605_Mark + new_p_1605_Mark;
        (…)
        p_2112_Mark = aux_p_2112_Mark + new_p_2112_Mark;

        //Save Input Signals
        last_inSignal_GO = inSignal_GO;
        (…)
```

```
    last_inSignal_GATEc3 = inSignal_GATEc3;

    // Reset Input Events
    inEvent_GO = 0;
    (…)
    inEvent_GATEc3 = 0;

    outSignal_Car3Moves = 0;
    (…)
    outSignal_Car3MovesBack = 0;

      if(p_1605_Mark >= 1)
       {
        if(outSignal_Car1Moves < 1)
          outSignal_Car1Moves = 1;
      }

     (…)

      if(p_1759_Mark >= 1)
       {
        if(outSignal_Car3MovesBack < 1)
          outSignal_Car3MovesBack = 1;
      }

    Analyse_Output_Events();

    return 0;
}
```

6.2. CONTROLADOR DISTRIBUÍDO

Se o objectivo for obter um controlador distribuído, é necessário dividir o modelo IOPT em três sub-modelos, cada um deles para ser implantado em um controlador local instalado em cada carro. Para atingir este objectivo, podemos utilizar a operação de divisão associada à ferramenta Split [Costa-Gomes, 09]. Na figura 6.4 estão identificados os nós por onde vai ser dividido o sistema. No caso da ferramenta de geração de código ANSI C esta utilização de um controlador distribuído permite exemplificar a tradução de eventos externos e remoção de sinais.

A aplicação do Split cria os três sub-modelos apresentados na figura 6.5.

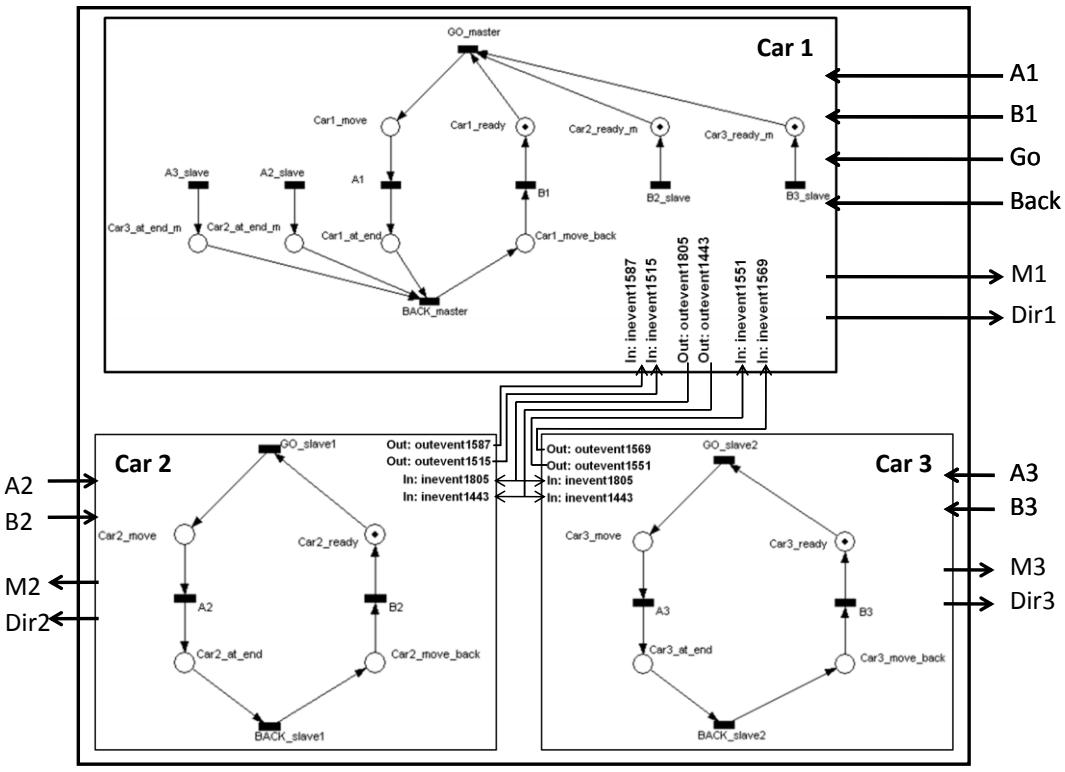

Figura 6.5 - Os três sub-modelos ligados por sinais de comunicação.

Tendo em conta que o sub-modelos do carro2 e carro3 são similares vai-se apenas apresentar o código gerado para o carro1 e para o carro3, o código do carro2 apenas terá o nome das variáveis como diferença para o código do carro3.

Para o carro1 a declaração das variáveis é mostrado abaixo.

```
//----------------Input Signals Values----------------------
char inSignal_GO;char last_inSignal_GO;
(…)
char inSignal_BACK;char last_inSignal_BACK;
//------------------------------------------------------------

//----------------Output Signals Values---------------------
```

```
char outSignal_Car1MovesBack;char last_outSignal_Car1MovesBack;
char outSignal_Car1Moves;char last_outSignal_Car1Moves;
//-----------------------------------------------------------

//----------------Input Events Values----------------------
char inEvent_GO;
(…)
char inevent1515;
//-----------------------------------------------------------

//----------------Output Events Values---------------------
char outevent1443;
char outevent1805;
//-----------------------------------------------------------

//---------------------Place Marks--------------------------
char p_18052684_Mark;char aux_p_18052684_Mark;char new_p_18052684_Mark;
(…)
char p_18053480_Mark;char aux_p_18053480_Mark;char new_p_18053480_Mark;
//-----------------------------------------------------------
```

No código abaixo está representada a função “start()” para o carro1

```
char start(void)
{
//----------------Init Input Signals Values----------------------
    inSignal_GO = 0;
    (…)
    inSignal_BACK = 0;
//-----------------------------------------------------------


//----------------Init Output Signals Values---------------------
    outSignal_Car1MovesBack = 0;
    outSignal_Car1Moves = 0;
//-----------------------------------------------------------

//----------------Init Input Events Values----------------------
    inEvent_GO = 0;
    (…)
    inevent1515 = 0;
//-----------------------------------------------------------

//----------------Init Output Events Values---------------------
    outevent1443 = 0;
    outevent1805 = 0;
//-----------------------------------------------------------

//----------------Init Place Marks----------------------
    p_18052684_Mark = 1; // Place Car3_ready_m
    (…)
    p_18053480_Mark = 0; // Place Car1_at_end
```

```
//-------------------------------------------------------

    return 0;
}
```

Para a execução de uma iteração na RdP do carro1 é criada a função “run()”. O seu código é mostrado abaixo.

```
char run(void)
{

    new_p_18052684_Mark = 0; // Place Car3_ready_m
    (…)
    new_p_18053480_Mark = 0; // Place Car1_at_end

    aux_p_18052684_Mark = p_18052684_Mark; // Place Car3_ready_m
    (…)
    aux_p_18053480_Mark = p_18053480_Mark; // Place Car1_at_end

    //Save Output Signals
    last_outSignal_Car1MovesBack = outSignal_Car1MovesBack;
    last_outSignal_Car1Moves = outSignal_Car1Moves;

    // Reset Output Events
    outevent1443 = 0;
    outevent1805 = 0;

    Analyse_Input_Events();


    // Transition 18052740 ( A2_m )
    if(inevent1587 == 1)
     {
       new_p_18052726_Mark = new_p_18052726_Mark + 1;
    }



    // Transition 18053494 ( B1 )
    if(aux_p_18053438_Mark >= 1 && inEvent_GATEc1 == 1)
     {
       aux_p_18053438_Mark = aux_p_18053438_Mark - 1;
       new_p_18053480_Mark = new_p_18053480_Mark + 1;
    }



    // Transition 18053530 ( B3_m )
    if(inevent1551 == 1)
```

```
     {
       new_p_18053466_Mark = new_p_18053466_Mark + 1;
     }

      (…)

     // Transition 18053566 ( B2_m )
     if(inevent1515 == 1)
      {
       new_p_18053452_Mark = new_p_18053452_Mark + 1;
     }

     // Actualize Marks
     p_18052684_Mark = aux_p_18052684_Mark + new_p_18052684_Mark;
     (…)
     p_18053480_Mark = aux_p_18053480_Mark + new_p_18053480_Mark;

     //Save Input Signals
     last_inSignal_GO = inSignal_GO;
    (…)
     last_inSignal_BACK = inSignal_BACK;

     // Reset Input Events
     inEvent_GO = 0;
     (…)
     inevent1515 = 0;

     outSignal_Car1MovesBack = 0;
     outSignal_Car1Moves = 0;

        if(p_18052712_Mark >= 1)
          {
            if(outSignal_Car1MovesBack < 1)
              outSignal_Car1MovesBack = 1;
        }
        if(p_18053438_Mark >= 1)
          {
            if(outSignal_Car1Moves < 1)
              outSignal_Car1Moves = 1;
        }


     Analyse_Output_Events();

     return 0;
}
```

Finalmente tem-se o carro3.

O código abaixo mostra a declaração das variáveis para o carro3.

```
//----------------Input Signals Values----------------------
char inSignal_GATEc3;char last_inSignal_GATEc3;
//----------------------------------------------------------

//----------------Output Signals Values---------------------
char outSignal_Car3Moves;char last_outSignal_Car3Moves;
char outSignal_Car3MovesBack;char last_outSignal_Car3MovesBack;
//----------------------------------------------------------

//----------------Input Events Values----------------------
char inEvent_GATEc3;
(…)
char inevent1805;
//----------------------------------------------------------

//----------------Output Events Values---------------------
char outevent1551;
char outevent1569;
//----------------------------------------------------------


//---------------------Place Marks-------------------------
char p_18057479_Mark;char aux_p_18057479_Mark;char new_p_18057479_Mark;
(…)
char p_18057959_Mark;char aux_p_18057959_Mark;char new_p_18057959_Mark;
//----------------------------------------------------------
```

A função de inicialização da rede, “start()” está representada no seguinte código.

```
char start(void)
{
//----------------Init Input Signals Values----------------------
    inSignal_GATEc3 = 0;
//----------------------------------------------------------

//----------------Init Output Signals Values----------------------
    outSignal_Car3Moves = 0;
    outSignal_Car3MovesBack = 0;
//----------------------------------------------------------

//----------------Init Input Events Values----------------------
    inEvent_GATEc3 = 0;
    (…)
    inevent1805 = 0;
//----------------------------------------------------------

//----------------Init Output Events Values----------------------
    outevent1551 = 0;
    outevent1569 = 0;
//----------------------------------------------------------
```

```
//----------------Init Place Marks---------------------
    p_18057479_Mark = 0; // Place Car3_move
    (…)
    p_18057959_Mark = 1; // Place Car3_ready
//-----------------------------------------------------

    return 0;
}
```

Para terminar, o código seguinte apresenta a função “run()” para a execução da RdP do carro3.

```
char run(void)
{

    new_p_18057479_Mark = 0; // Place Car3_move
    (…)
    new_p_18057959_Mark = 0; // Place Car3_ready

    aux_p_18057479_Mark = p_18057479_Mark; // Place Car3_move
    (…)
    aux_p_18057959_Mark = p_18057959_Mark; // Place Car3_ready

    //Save Output Signals
    last_outSignal_Car3Moves = outSignal_Car3Moves;
    last_outSignal_Car3MovesBack = outSignal_Car3MovesBack;

    // Reset Output Events
    outevent1551 = 0;
    outevent1569 = 0;

    Analyse_Input_Events();


    // Transition 18057543 ( B3 )
    if(aux_p_18057479_Mark >= 1 && inEvent_GATEc3 == 1)
     {
       aux_p_18057479_Mark = aux_p_18057479_Mark - 1;
       new_p_18057493_Mark = new_p_18057493_Mark + 1;
       outevent1551 = 1;
     }

      (…)

    // Transition 18058009 ( BACK_m4 )
    if(aux_p_18057493_Mark >= 1 && inevent1805 == 1)
     {
       aux_p_18057493_Mark = aux_p_18057493_Mark - 1;
       new_p_18057945_Mark = new_p_18057945_Mark + 1;
     }
```

```
    // Actualize Marks
    p_18057479_Mark = aux_p_18057479_Mark + new_p_18057479_Mark;
    (…)
    p_18057959_Mark = aux_p_18057959_Mark + new_p_18057959_Mark;

    //Save Input Signals
    last_inSignal_GATEc3 = inSignal_GATEc3;

    // Reset Input Events
    inEvent_GATEc3 = 0;
    (…)
    inevent1805 = 0;

    outSignal_Car3Moves = 0;
    outSignal_Car3MovesBack = 0;

       if(p_18057479_Mark >= 1)
        {
          if(outSignal_Car3Moves < 1)
            outSignal_Car3Moves = 1;
       }
       if(p_18057945_Mark >= 1)
        {
          if(outSignal_Car3MovesBack < 1)
            outSignal_Car3MovesBack = 1;
       }
    Analyse_Output_Events();

    return 0;
}
```

CONCLUSÕES E TRABALHO FUTURO

As RdP mostram-se, quando devidamente suportadas por um conjunto de ferramentas adequadas, como um formalismo bastante eficiente na modelação, análise, simulação e até implementação de sistemas.

As RdP IOPT, por permitirem a comunicação com o meio exterior permitem um avanço nesta implementação pois permitem modular a comunicação com o exterior.

Com esta dissertação deu-se mais um passo na disponibilização das RdP IOPT como ferramenta eficiente na modelação e implementação de sistemas acrescentando uma ferramenta que permite a sua tradução automática para a linguagem ANSI C.

Mostrou-se que é possível a geração automática de código ANSI C válido que permite executar a rede de forma a garantir a semântica de execução pretendida, e tornar possível programar, por exemplo, microcontroladores directamente através de RdP IOPT.

A ferramenta gerada permite a tradução de ficheiros PNML para controladores centralizados e também para controladores distribuídos, permitindo a comunicação de eventos externos. Desta forma é possível a integração com o PNML gerado pelas ferramentas do projecto FORDESIGN tais como o Split.

O código gerado tem velocidade e dimensão (tanto em numero de linhas como em espaço ocupado) semelhantes ao código criado manualmente de forma específica para cada sistema.

Foram realizados testes do código gerado com sucesso através da sua integração em projectos desenvolvidos no âmbito de outras dissertações de mestrado, nomeadamente de controladores distribuídos obtidos a partir da partição de modelos IOPT e interligados a Network-on-Chip, bem como a controladores distribuídos para controlo domótico (TinyDomots).

A introdução do novo meta-modelo das RdP IOPT introduziu muitas mais-valias para a modelação e implementação de sistemas e criação de ferramentas para o seu uso.

Como trabalho futuro seria interessante um estudo mais aprofundado quanto ao novo meta-modelo das RdP IOPT de forma a determinar formas de simplificar e reduzir a dimensão do código ANSI C gerado.

REFERÊNCIAS

[ANSI 89] American National Standards Institute, American National Standard for Information Systems-Programming Language C, X3.159-1989.

[Barros, 96] Barros, João Paulo M. P. Ramos, “CpPNeTS: Uma Classe de Redes de Petri de Alto-nível Implementação de um sistema de suporte à sua aplicação e análise”, dissertação apresentada na Faculdade de Ciências e Tecnologia da Universidade Nova de Lisboa, como parte dos requisitos necessários à obtenção do grau de Mestre em Engenharia Informática, Lisboa, 1996.

[Barros-Gomes, 04] João Paulo Barros, Luís Gomes, " Operational PNML: Towards a PNML Support for Model Construction and Modification”, Workshop on the Definition, Implementation and Application of a Standard Interchange Format for Petri Nets, Satellite event at the International Conference on Application and Theory of Petri Nets 2004, 2004.

[Brauer-Reisig, 06] W. Brauer e W. Reisig, “Carl Adam Petri and ‘Petri Nets’”, 2006. http://www.informatik.uni-hamburg.de/TGI/PetriNets/history/, Acedido em Janeiro 2009.

[Conceição, 04] Conceição, Paulo Alexandre Meira da, “De Casos de Uso a Redes de Petri: Uma aplicação à monitorização de edifícios”,

Dissertação de Mestrado em Engenharia Informática na Faculdade de Ciências e Tecnologia da Universidade Nova de Lisboa, 2004.

[Costa-Gomes, 09] A. Costa, L. Gomes; “Petri net partitioning using net splitting operation”; INDIN’2009 - 7th IEEE International Conference on Industrial Informatics, 24-26 June 2009, Cardiff, UK; DOI 10.1109/INDIN.2009.5195804

[David-Alla, 09] R. David, H. Alla, Petri Nets and Grafcet Tools for modeling discrete event systems, Prentice Hall, 1992.

[DocBook, 10] Overview of DocBook, http://www.docbook.org/, Acedido em Junho 2010

[FORDESIGN, 07] FORDESIGN – Formal Methods for Embedded Systems Co-Design. 2007. R&D Project POSC/EIA/61364/2004 com o apoio FCT - Fundação para a Ciência e a Tecnologia. http://www.uninova.pt/fordesign/, Acedido em , 2010.

[Gomes, 97] Gomes, Luís Filipe dos Santos, “Redes de Petri Reactivas e Hierárquicas – Integração de Formalismos no Projecto de Sistema Reactivos de Tempo Real”, Tese de Doutoramento em Engenharia Electrotécnica na Faculdade de Ciências e Tecnologia da Universidade Nova de Lisboa, 1997.

[Gomes-Barros, 03] L. Gomes, J. P. Barros, “On structuring mechanisms for Petri nets based system design”, in Proc. 2003 IEEE Conf.

Emerging Technologies and Factory Automation (ETFA 2003), Sep. 2003, pp. 431-438.

[Gomes et al, 06] Luís Gomes, João Paulo Barros e Anikó Costa, “Modeling Formalisms for Embedded Systems”, The Industrial Information Technology Handbook, Richard Zurawski, Section I, Cap. 5, CRC, ISBN 0849328241, 9780849328244, 2006.

[Gomes et al, 07] Luís Gomes, João Paulo Barros, Anikó Costa, Ricardo Nunes "The Input-Output Place-Transition Petri Net Class and Associated Tools", Industrial Informatics, 2007 5th IEEE International Conference, Vol. 1, Págs: 509 - 514, 2007

[Gomes et al, 10] Luís Gomes, Rogério Rebelo, João Paulo Barros, Anikó Costa, Rui Pais; “From Petri net models to C implementation of digital controllers”; ISIE’2010 - IEEE International Symposium on Industrial Electronics, 4-7 July 2010, Bari, Italy;

[Heuser, 91] Heuser, Modelação Conceitual de Sistemas. (1a. Edição, EBAI - 1988) 2a. Edição, Campinas: UNICAMP (IV Escola Brasileiro-Argentina de Informática), 1991.

[HTML, 10] HTML home page, http://wwww3.org/html/, Acedido em Junho 2010

[Jensen, 97] Jensen, Kurt, "A brief introduction to Coloured Petri Nets", Lecture Notes in Computer Science, Tools and Algorithms for

the Construction and Analysis of Systems, Págs. 203-208, Springer-Verlag, ISBN:3-540-62790-1, 1997.

[Jensen, 92-97] K. Jensen, Colored Petri Nets, Basic Concepts, Analysis Methods and Practical Use – Volumes 1-3 Berlin, Germany: Springer-Verlag, 1992-1997.

[Johnson 79a] S. C. Johnson, `Yet another compiler-compiler,' in Unix Programmer's Manual, Seventh Edition, Vol. 2A, M. D. McIlroy and B. W. Kernighan, eds. AT&T Bell Laboratories: Murray Hill, NJ, 1979.

[Johnson 79b] S. C. Johnson, `Lint, a Program Checker,' in Unix Programmer's Manual, Seventh Edition, Vol. 2B, M. D. McIlroy and B. W. Kernighan, eds. AT&T Bell Laboratories: Murray Hill, NJ, 1979.

[Jüngel et.al, 00] Matthias Jüngel, Ekkart Kindler, Michael Weber "The Petri Net Markup Language", Workshop on Algorithms and Tools for Petri Nets, 2000.

[Kernighan 78] B. W. Kernighan and D. M. Ritchie, The C Programming Language, Prentice-Hall: Englewood Cliffs, NJ, 1978. Second edition, 1988.

[Kindler -Weber, 01] Ekkart Kindler and Michael Weber. A universal module concept for Petri nets. In Proceedings des 8.Workshops Algorithmen und Werkzeuge fr Petrinetze / Gabriel Juhas und Robert Lorenz (Hrsg.) -Katholischen Universit□at Eichst□att, 2001, pag. 7-12, 1-2 Outubro 2001.

[Lesk 73] M. E. Lesk, “A Portable I/O Package,” AT&T Bell Laboratories internal memorandum ca. 1973.

[Lino, 03] Lino, Rui Manuel Gonçalves, “Detecção de Falhas em Sistemas de Automação utilizando Redes de Petri”, Dissertação de Mestrado no Departamento de Departamento de Ciências dos Materiais na Faculdade de Ciências e Tecnologia da Universidade Nova de Lisboa, 2003.

[Manzano, 04] José Navarro Manzano, “Revisão e Discussão da Norma ISO 5807 - 1985” Faculdade Cantareira, São Paulo, 2004. http://www.manzano.pro.br/alunado/navarro.pdf, Acedido em Julho 2010.

[Mealy, 55] G. H. Mealy, “Method for Synthesizing Sequential Circuits” Bell System Technical Journal, 34, 1955, pg. 1045-1079.

[Moore, 56] E. Moore, “Gedanken-Experiments on Sequential Machines”, Automata Studies. New Jersey, Princeton 1956.

[Moen, 03] Moen, Anders, “Introduction to Petri Nets”, 2003, http://www.ifi.uio.no/dbsem/2003vaar.html, Acedido em Janeiro 2009

[Murata, 89] Murata, Tadao, “Petri Nets: Properties, Analysis and Applications”, Proceedings of IEEE, Vol. 77-4, 1989.

[Organick, 75] E. I. Organick, The Multics System: An Examination of its Structure, MIT Press: Cambridge, Mass., 1975.

[Pais, 04] Pais, Rui Manuel Carvalho, “Geração de Executores e Analisadores de Redes de Petri”, Dissertação de Mestrado no Departamento de Engenharia Informática na Faculdade de Ciências e Tecnologia da Universidade Nova de Lisboa, 2004.

[PEP, 10] PEP Homepage, http://peptool.sourceforge.net/, Acedido em Junho 2010.

[Peterson, 77] Peterson, James, ”Petri Nets”, ACM Computing Surveys, Vol. 9-3, Págs. 223-252, 1977.

[Petri Nets World, 10] Petri Nets World, http://www.informatik.uni-hamburg.de/TGI/PetriNets/, Acedido em Junho2010.

[Petri, 62] Carl Adam Petri, Dissertação de doutoramento, “Kommunikation mit Automaten (Communication with Automata)”, 1962.

[Petri, 63] Fundamentals of a Theory of Asynchronous Information Flow. 1st IFIP World Computer Congress. Munich 1962, North Holland, pp 386-390, 1963.

[Petri, 67] “Grundsätzliches zur Beschreibung diskreter Prozesse (Fundamentals on the description of discrete processes)”. 3rd Colloquium on Automata Theory in Hannover, Birkhäuser-Verlag, pp121-140, 1967.

[Petrify, 10] Petrify Homepage, http://www.lsi.upc.es/~jordicf/petrify/home.html, Acedido em Julho 2010.

[PNML, 04] Petri Net Markup Language, http://www.informatik.huberlin.de/top/pnml/about.html/, acedido em Julho 2010.

[Reis, 08] Reis,Tiago Miguel Correia, “Partição de modelos de Redes de Petri”, Dissertação de Mestrado em Engenharia de Electrotécnica e de Computadores na Faculdade de Ciências e Tecnologia da Universidade Nova de Lisboa, 2004.

[Richards, 67] M. Richards, “The BCPL Reference Manual”, MIT Project MAC Memorandum M-352, July 1967.

[Ritchie 78] D. M. Ritchie, “UNIX: A Retrospective”, Bell Sys. Tech. J. 57 (6) (part 2), July-Aug, 1978.

[Ritchie 84] D. M. Ritchie, `The Evolution of the UNIX Time-sharing System,' AT&T Bell Labs. Tech. J. 63 (8) (part 2), Oct. 1984.

[SGML, 04] SGML Homepage, http://www.w3.org/MarkUp/SGML/, 2004, acedido em Julho 2010.

[Sgroi, 00] M. Sgroi, L.Lavagno e A. Sangiovanni-Vincentelli, “Formal Models for Embedded Systems Design”, IEEE Design and Test of Computers, Vol. 17, Págs. 14-17, 2000.

[Silva, 85] M. Silva, Las Redes de Petri: en la Automática y la Informática.Madrid: Editorial AC, 1985.

[SIPN, 10] SIPN Editor, http://www.eit.uni-kl.de/litz/ENGLISH/software/SIPNEditor.htm, acedido em Julho 2010.

[Weber-Kindler, 02] Michael Weber, Ekkard Kindler "The Petri Net Markup Language", LNCS Petri Net Technology for Communication-Based Systems, ISBN 978-3-540-20538-8, 2002.

[Weber-Kindler, 03] Michael Weber and Ekkart Kindler. The Petri net markup language. In H. Ehrig, W. Reisig, G. Rozenberg, and H. Weber, editors, Petri Net Technology for Comunication Based Systems, volume 2472 of LNCS, pag. 124-144. Springer-Verlag, 2003.

[Wegrzyn et al, 97] Marek Wegrzyn, Pawel Wolanski, Marian Adamski, Joao L. Monteiro, “Coloured Petri Net Model of Application Specific Logic Controller Programs”, Industrial Electronics, ISIE '97, Proceedings of the IEEE International Symposium, Vol. 1, 7-11 p.158 - 163, 1997.

[XML, 03] XML – Extensible Markup Language, http://www.w3.org/XML/, 2003, acedido em Julho 2010.

[Zurawski et al, 94] Zurawski, R.; MengChu Zhou, “Petri nets and industrial applications: A tutorial”, Industrial Electronics, IEEE Transactions, Vol. 41, Págs. 567-583, 1994.

ANEXO 1 -MANUAL DE UTILIZAÇÃO DA FERRAMENTA GRÁFICA.

O interface gráfico inicia com uma janela como a apresenta da na figura A1.0.1. Nesta janela o utilizador tem a possibilidade de carregar um ficheiro PNML.

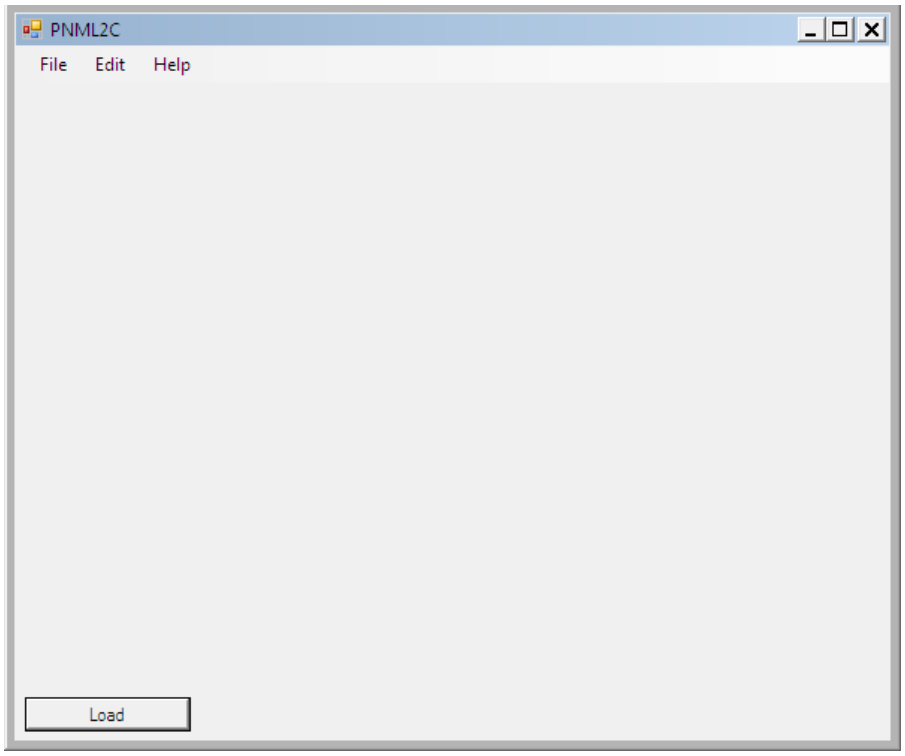

Figura A1.0.1 - Janela inicial da ferramenta.

Depois de carregar no botão de carregamento do ficheiro (“Load”) aparece uma janela para escolher o ficheiro, como a apresentada na figura A1.0.2.

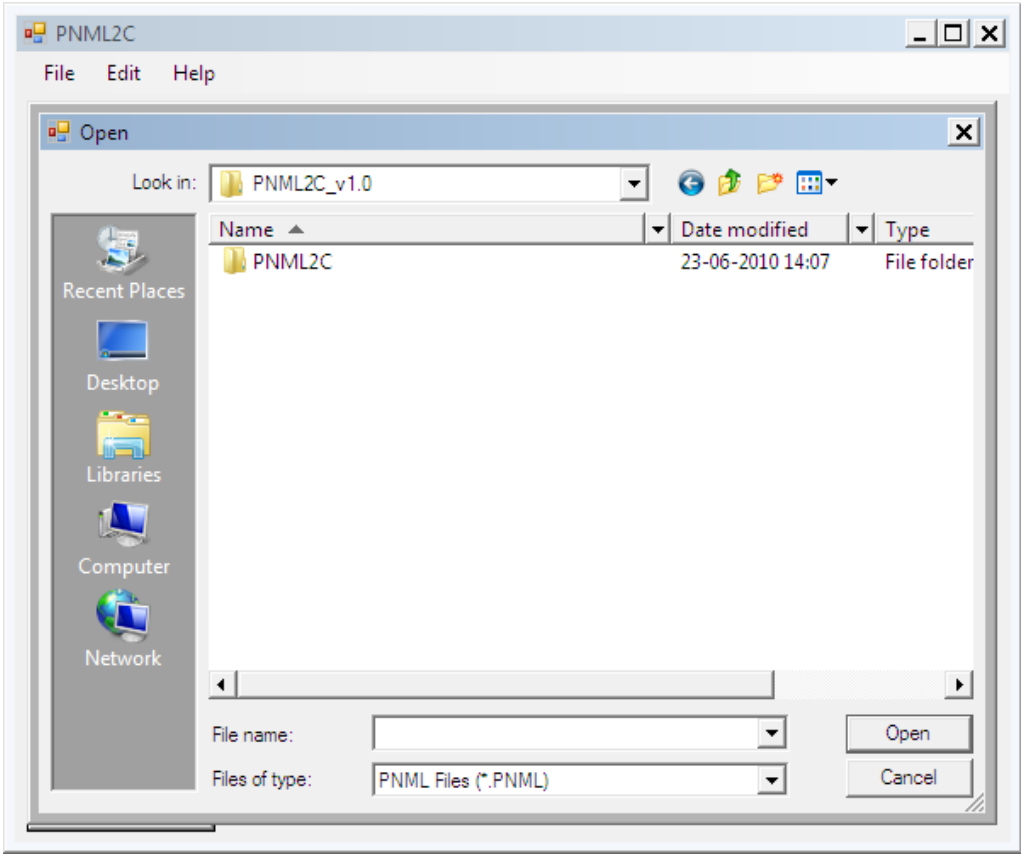

Figura A1.0.2 - Carregamento de um ficheiro PNML na ferramenta.

Nesta altura aparece uma janela onde se encontram listas dos sinais de entrada e de saída e dos eventos de entrada e de saída. Todos eles têm uma “checkBox” marcada. O utilizador deve desmarcar os sinais que não quer ver representados no código C e os eventos que vão ser externos, ou seja, o seu valor vem do exterior. Nesta janela tem também um botão que permite gerar o código, como mostra a figura A1.0.3.

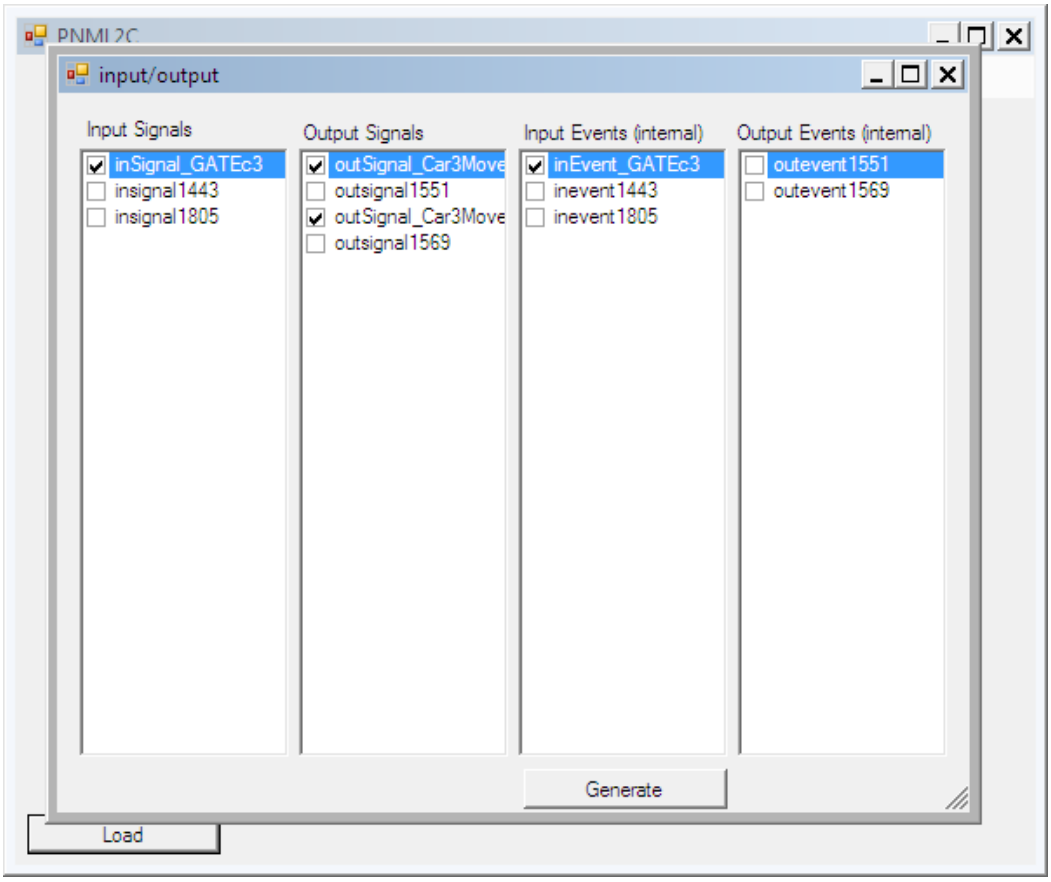

Figura A1.0.3 - Selecção de eventos internos e externos.

Depois de carregado, o programa gera automaticamente o código dos cinco ficheiros ANSI C já referidos. Como mostra a figura A1.0.4, o PNML e o código dos ficheiros gerados são apresentados em separado de forma a poderem ser consultados pelo utilizador antes de serem salvos.

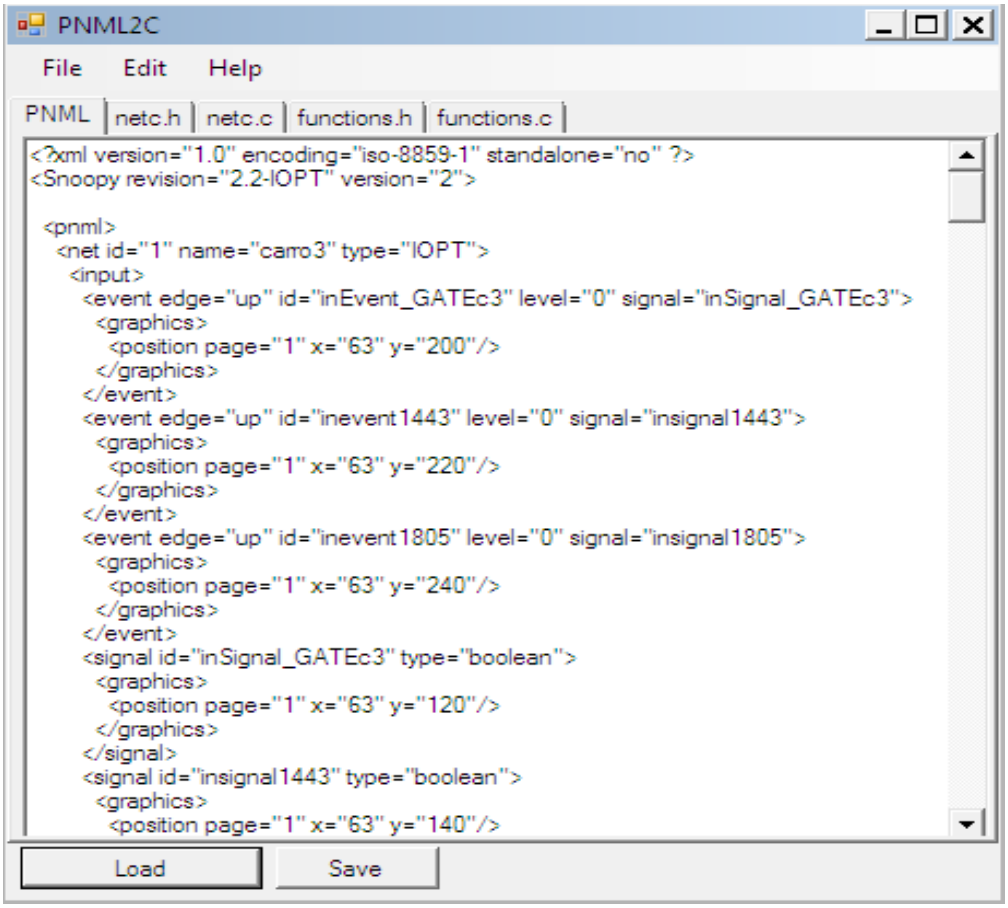

Figura A1.0.4 - Apresentação do código gerado.

Como se mostra também na figura A1.0.4 está disponivel um botão que permite salvar os ficheiros do projecto gerado.

Ao carregar neste botão é apresentada uma janela, mostrada na figura A1.0.5, que permite escolher a pasta onde salvar o projecto em ANSI C criado. Nessa pasta escolhida pelo utilizador é criada uma nova pasta com o nome da RdP definido no PNML. Dentro desta pasta são salvos os cinco ficheiros do projecto.

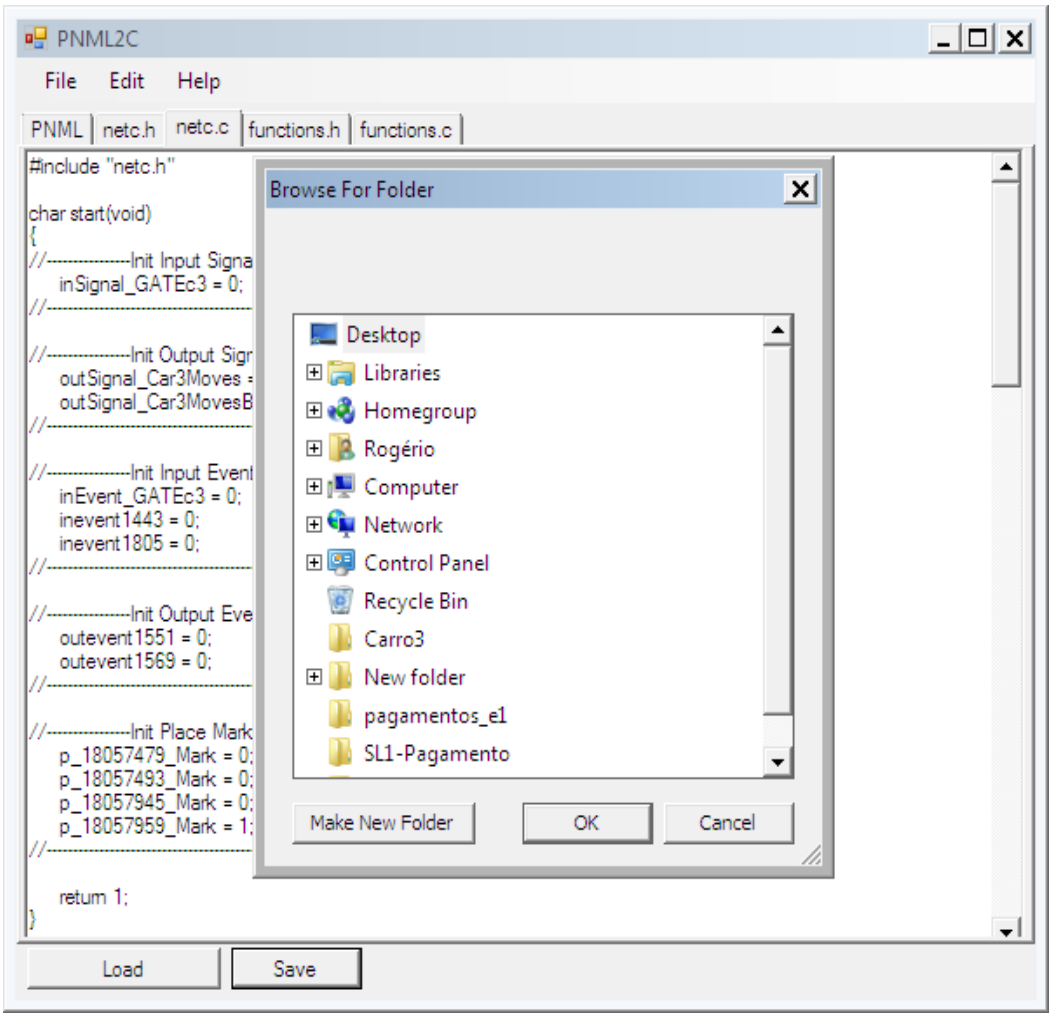

Figura A1.0.5 - Escolha da pasta onde salvar os ficheiros gerados.

Em seguida o programa volta à sua forma inicial, permitindo ser fechado ou voltar a ser usado para mais um ficheiro PNML.

Neste momento o código continua disponível também caso se queira voltar a guardar voltando a carregar no botão “Save”.

ANEXO 2 – FICHEIROS GERADOS PARA OS 3 CARROS

MAIN.C

```
#include <string.h>
#include "ioptc.h"

//----------------Input Signals Values----------------------
char inSignal_GO;char last_inSignal_GO;
char inSignal_BACK;char last_inSignal_BACK;
char inSignal_GATEc1;char last_inSignal_GATEc1;
char inSignal_GATEc2;char last_inSignal_GATEc2;
char inSignal_GATEc3;char last_inSignal_GATEc3;
//----------------------------------------------------------

//----------------Output Signals Values----------------------
char outSignal_Car3Moves;char last_outSignal_Car3Moves;
char outSignal_Car1Moves;char last_outSignal_Car1Moves;
char outSignal_Car2Moves;char last_outSignal_Car2Moves;
char outSignal_Car2MovesBack;char last_outSignal_Car2MovesBack;
char outSignal_Car1MovesBack;char last_outSignal_Car1MovesBack;
char outSignal_Car3MovesBack;char last_outSignal_Car3MovesBack;
//----------------------------------------------------------

//----------------Input Events Values----------------------
char inEvent_GO;
char inEvent_BACK;
char inEvent_GATEc1;
char inEvent_GATEc2;
char inEvent_GATEc3;
//----------------------------------------------------------

//----------------Output Events Values----------------------
char outEvent_Car2Moves;
char outEvent_Car1Moves;
char outEvent_Car3Moves;
char outEvent_Car1MovesBack;
char outEvent_Car2MovesBack;
char outEvent_Car3MovesBack;
//----------------------------------------------------------

//---------------------Place Marks-------------------------
char p_1605_Mark;char aux_p_1605_Mark;char new_p_1605_Mark;
char p_1619_Mark;char aux_p_1619_Mark;char new_p_1619_Mark;
char p_1633_Mark;char aux_p_1633_Mark;char new_p_1633_Mark;
char p_1647_Mark;char aux_p_1647_Mark;char new_p_1647_Mark;
char p_1661_Mark;char aux_p_1661_Mark;char new_p_1661_Mark;
char p_1675_Mark;char aux_p_1675_Mark;char new_p_1675_Mark;
char p_1703_Mark;char aux_p_1703_Mark;char new_p_1703_Mark;
char p_1717_Mark;char aux_p_1717_Mark;char new_p_1717_Mark;
char p_1731_Mark;char aux_p_1731_Mark;char new_p_1731_Mark;
char p_1745_Mark;char aux_p_1745_Mark;char new_p_1745_Mark;
```

```
char p_1759_Mark;char aux_p_1759_Mark;char new_p_1759_Mark;
char p_2112_Mark;char aux_p_2112_Mark;char new_p_2112_Mark;
//-------------------------------------------------------


int main()
{
  start();

  //insert your code here...

  //Set Input Signals
  //Set Input Events

  run();

  //Get Output Signals
  //Get Output Events

  //insert your code here...

  return 0;
}
```

FUNCTIONS.H

```
#include <string.h>
#include <stdio.h>
#include <stdlib.h>

//----------------Input Signals Values----------------------
extern char inSignal_GO;extern char last_inSignal_GO;
extern char inSignal_BACK;extern char last_inSignal_BACK;
extern char inSignal_GATEc1;extern char last_inSignal_GATEc1;
extern char inSignal_GATEc2;extern char last_inSignal_GATEc2;
extern char inSignal_GATEc3;extern char last_inSignal_GATEc3;
//-------------------------------------------------------

//----------------Output Signals Values----------------------
extern char outSignal_Car3Moves;extern char last_outSignal_Car3Moves;
extern char outSignal_Car1Moves;extern char last_outSignal_Car1Moves;
extern char outSignal_Car2Moves;extern char last_outSignal_Car2Moves;
extern char outSignal_Car2MovesBack;extern char last_outSignal_Car2MovesBack;
extern char outSignal_Car1MovesBack;extern char last_outSignal_Car1MovesBack;
extern char outSignal_Car3MovesBack;extern char last_outSignal_Car3MovesBack;
//-------------------------------------------------------

//----------------Input Events Values----------------------
extern char inEvent_GO;
extern char inEvent_BACK;
```

```
extern char inEvent_GATEc1;
extern char inEvent_GATEc2;
extern char inEvent_GATEc3;
//-----------------------------------------------------------

//----------------Output Events Values---------------------
extern char outEvent_Car2Moves;
extern char outEvent_Car1Moves;
extern char outEvent_Car3Moves;
extern char outEvent_Car1MovesBack;
extern char outEvent_Car2MovesBack;
extern char outEvent_Car3MovesBack;
//-----------------------------------------------------------

//----------------Place Marks----------------------
extern char p_1605_Mark;extern char aux_p_1605_Mark;extern char new_p_1605_Mark;
extern char p_1619_Mark;extern char aux_p_1619_Mark;extern char new_p_1619_Mark;
extern char p_1633_Mark;extern char aux_p_1633_Mark;extern char new_p_1633_Mark;
extern char p_1647_Mark;extern char aux_p_1647_Mark;extern char new_p_1647_Mark;
extern char p_1661_Mark;extern char aux_p_1661_Mark;extern char new_p_1661_Mark;
extern char p_1675_Mark;extern char aux_p_1675_Mark;extern char new_p_1675_Mark;
extern char p_1703_Mark;extern char aux_p_1703_Mark;extern char new_p_1703_Mark;
extern char p_1717_Mark;extern char aux_p_1717_Mark;extern char new_p_1717_Mark;
extern char p_1731_Mark;extern char aux_p_1731_Mark;extern char new_p_1731_Mark;
extern char p_1745_Mark;extern char aux_p_1745_Mark;extern char new_p_1745_Mark;
extern char p_1759_Mark;extern char aux_p_1759_Mark;extern char new_p_1759_Mark;
extern char p_2112_Mark;extern char aux_p_2112_Mark;extern char new_p_2112_Mark;
//-----------------------------------------------------------



//Input Signals
void Set_inSignal_GO (char val);
void Set_inSignal_BACK (char val);
void Set_inSignal_GATEc1 (char val);
void Set_inSignal_GATEc2 (char val);
void Set_inSignal_GATEc3 (char val);


//Output Signal
char Get_outSignal_Car3Moves (void);
char Get_outSignal_Car1Moves (void);
char Get_outSignal_Car2Moves (void);
char Get_outSignal_Car2MovesBack (void);
char Get_outSignal_Car1MovesBack (void);
char Get_outSignal_Car3MovesBack (void);


//Input Event
void Set_inEvent_GO (char val);
void Set_inEvent_BACK (char val);
void Set_inEvent_GATEc1 (char val);
void Set_inEvent_GATEc2 (char val);
void Set_inEvent_GATEc3 (char val);
```

```
//Output Event
char Get_outEvent_Car2Moves (void);
char Get_outEvent_Car1Moves (void);
char Get_outEvent_Car3Moves (void);
char Get_outEvent_Car1MovesBack (void);
char Get_outEvent_Car2MovesBack (void);
char Get_outEvent_Car3MovesBack (void);

void Analyse_Input_Events(void);
void Analyse_Output_Events(void);
```

FUNCTIONS.C

```
#include "functions.h"

//Input Signal
void Set_inSignal_GO (char val)
{
  last_inSignal_GO = inSignal_GO;
  inSignal_GO = val;
}

//Input Signal
void Set_inSignal_BACK (char val)
{
  last_inSignal_BACK = inSignal_BACK;
  inSignal_BACK = val;
}

//Input Signal
void Set_inSignal_GATEc1 (char val)
{
  last_inSignal_GATEc1 = inSignal_GATEc1;
  inSignal_GATEc1 = val;
}

//Input Signal
void Set_inSignal_GATEc2 (char val)
{
  last_inSignal_GATEc2 = inSignal_GATEc2;
  inSignal_GATEc2 = val;
}

//Input Signal
void Set_inSignal_GATEc3 (char val)
{
  last_inSignal_GATEc3 = inSignal_GATEc3;
  inSignal_GATEc3 = val;
```

```
}


//Output Signal
char Get_outSignal_Car3Moves (void)
{
  return outSignal_Car3Moves;
}

//Output Signal
char Get_outSignal_Car1Moves (void)
{
  return outSignal_Car1Moves;
}

//Output Signal
char Get_outSignal_Car2Moves (void)
{
  return outSignal_Car2Moves;
}

//Output Signal
char Get_outSignal_Car2MovesBack (void)
{
  return outSignal_Car2MovesBack;
}

//Output Signal
char Get_outSignal_Car1MovesBack (void)
{
  return outSignal_Car1MovesBack;
}

//Output Signal
char Get_outSignal_Car3MovesBack (void)
{
  return outSignal_Car3MovesBack;
}


//Input Event
void Set_inEvent_GO (char val)
{
  inEvent_GO = val;
}

//Input Event
void Set_inEvent_BACK (char val)
{
  inEvent_BACK = val;
}

//Input Event
void Set_inEvent_GATEc1 (char val)
```

```
{
  inEvent_GATEc1 = val;
}

//Input Event
void Set_inEvent_GATEc2 (char val)
{
  inEvent_GATEc2 = val;
}

//Input Event
void Set_inEvent_GATEc3 (char val)
{
  inEvent_GATEc3 = val;
}


//Output Event
char Get_outEvent_Car2Moves (void)
{
  return outEvent_Car2Moves;
}

//Output Event
char Get_outEvent_Car1Moves (void)
{
  return outEvent_Car1Moves;
}

//Output Event
char Get_outEvent_Car3Moves (void)
{
  return outEvent_Car3Moves;
}

//Output Event
char Get_outEvent_Car1MovesBack (void)
{
  return outEvent_Car1MovesBack;
}

//Output Event
char Get_outEvent_Car2MovesBack (void)
{
  return outEvent_Car2MovesBack;
}

//Output Event
char Get_outEvent_Car3MovesBack (void)
{
  return outEvent_Car3MovesBack;
}

void Analyse_Input_Events(void)
```

```
{
  if(inSignal_GO == 1 && last_inSignal_GO == 0)
     inEvent_GO = 1;
  if(inSignal_BACK == 1 && last_inSignal_BACK == 0)
     inEvent_BACK = 1;
  if(inSignal_GATEc1 == 1 && last_inSignal_GATEc1 == 0)
     inEvent_GATEc1 = 1;
  if(inSignal_GATEc2 == 1 && last_inSignal_GATEc2 == 0)
     inEvent_GATEc2 = 1;
  if(inSignal_GATEc3 == 1 && last_inSignal_GATEc3 == 0)
     inEvent_GATEc3 = 1;
}

void Analyse_Output_Events(void)
{
  if(outSignal_Car2Moves == 1 && last_outSignal_Car2Moves == 0)
     outEvent_Car2Moves = 1;
  if(outSignal_Car1Moves == 1 && last_outSignal_Car1Moves == 0)
     outEvent_Car1Moves = 1;
  if(outSignal_Car3Moves == 1 && last_outSignal_Car3Moves == 0)
     outEvent_Car3Moves = 1;
  if(outSignal_Car1MovesBack == 1 && last_outSignal_Car1MovesBack == 0)
     outEvent_Car1MovesBack = 1;
  if(outSignal_Car2MovesBack == 1 && last_outSignal_Car2MovesBack == 0)
     outEvent_Car2MovesBack = 1;
  if(outSignal_Car3MovesBack == 1 && last_outSignal_Car3MovesBack == 0)
     outEvent_Car3MovesBack = 1;
}
```

IOPTC.H

```
#include <string.h>
#include <stdio.h>
#include <stdlib.h>
#include "functions.h"


//----------------Input Signals Values----------------------
extern char inSignal_GO;extern char last_inSignal_GO;
extern char inSignal_BACK;extern char last_inSignal_BACK;
extern char inSignal_GATEc1;extern char last_inSignal_GATEc1;
extern char inSignal_GATEc2;extern char last_inSignal_GATEc2;
extern char inSignal_GATEc3;extern char last_inSignal_GATEc3;
//----------------------------------------------------------

//----------------Output Signals Values----------------------
extern char outSignal_Car3Moves;extern char last_outSignal_Car3Moves;
extern char outSignal_Car1Moves;extern char last_outSignal_Car1Moves;
extern char outSignal_Car2Moves;extern char last_outSignal_Car2Moves;
```

```
extern char outSignal_Car2MovesBack;extern char last_outSignal_Car2MovesBack;
extern char outSignal_Car1MovesBack;extern char last_outSignal_Car1MovesBack;
extern char outSignal_Car3MovesBack;extern char last_outSignal_Car3MovesBack;
//-----------------------------------------------------------

//----------------Input Events Values---------------------
extern char inEvent_GO;
extern char inEvent_BACK;
extern char inEvent_GATEc1;
extern char inEvent_GATEc2;
extern char inEvent_GATEc3;
//-----------------------------------------------------------

//----------------Output Events Values---------------------
extern char outEvent_Car2Moves;
extern char outEvent_Car1Moves;
extern char outEvent_Car3Moves;
extern char outEvent_Car1MovesBack;
extern char outEvent_Car2MovesBack;
extern char outEvent_Car3MovesBack;
//-----------------------------------------------------------


//----------------Place Marks---------------------
extern char p_1605_Mark;extern char aux_p_1605_Mark;extern char new_p_1605_Mark;
extern char p_1619_Mark;extern char aux_p_1619_Mark;extern char new_p_1619_Mark;
extern char p_1633_Mark;extern char aux_p_1633_Mark;extern char new_p_1633_Mark;
extern char p_1647_Mark;extern char aux_p_1647_Mark;extern char new_p_1647_Mark;
extern char p_1661_Mark;extern char aux_p_1661_Mark;extern char new_p_1661_Mark;
extern char p_1675_Mark;extern char aux_p_1675_Mark;extern char new_p_1675_Mark;
extern char p_1703_Mark;extern char aux_p_1703_Mark;extern char new_p_1703_Mark;
extern char p_1717_Mark;extern char aux_p_1717_Mark;extern char new_p_1717_Mark;
extern char p_1731_Mark;extern char aux_p_1731_Mark;extern char new_p_1731_Mark;
extern char p_1745_Mark;extern char aux_p_1745_Mark;extern char new_p_1745_Mark;
extern char p_1759_Mark;extern char aux_p_1759_Mark;extern char new_p_1759_Mark;
extern char p_2112_Mark;extern char aux_p_2112_Mark;extern char new_p_2112_Mark;
//-----------------------------------------------------------

char start(void);
char run(void);
```

IOPTC.C

```
#include "ioptc.h"

char start(void)
{
//----------------Init Input Signals Values----------------------
      inSignal_GO = 0;
      inSignal_BACK = 0;
      inSignal_GATEc1 = 0;
      inSignal_GATEc2 = 0;
      inSignal_GATEc3 = 0;
//-----------------------------------------------------------

//----------------Init Output Signals Values----------------------
      outSignal_Car3Moves = 0;
      outSignal_Car1Moves = 0;
      outSignal_Car2Moves = 0;
      outSignal_Car2MovesBack = 0;
      outSignal_Car1MovesBack = 0;
      outSignal_Car3MovesBack = 0;
//-----------------------------------------------------------

//----------------Init Input Events Values----------------------
      inEvent_GO = 0;
      inEvent_BACK = 0;
      inEvent_GATEc1 = 0;
      inEvent_GATEc2 = 0;
      inEvent_GATEc3 = 0;
//-----------------------------------------------------------

//----------------Init Output Events Values----------------------
      outEvent_Car2Moves = 0;
      outEvent_Car1Moves = 0;
      outEvent_Car3Moves = 0;
      outEvent_Car1MovesBack = 0;
      outEvent_Car2MovesBack = 0;
      outEvent_Car3MovesBack = 0;
//-----------------------------------------------------------

//----------------Init Place Marks----------------------
      p_1605_Mark = 0; // Place Car1_move
      p_1619_Mark = 0; // Place Car2_move
      p_1633_Mark = 0; // Place Car3_move
      p_1647_Mark = 1; // Place Car1_ready
      p_1661_Mark = 1; // Place Car2_ready
      p_1675_Mark = 1; // Place Car3_ready
      p_1703_Mark = 0; // Place Car2_at_end
```

```
    p_1717_Mark = 0; // Place Car1_at_end
    p_1731_Mark = 0; // Place Car1_move_back
    p_1745_Mark = 0; // Place Car2_move_back
    p_1759_Mark = 0; // Place Car3_move_back
    p_2112_Mark = 0; // Place Car3_at_end
//-----------------------------------------------------------

    return 0;
}

char run(void)
{

    new_p_1605_Mark = 0; // Place Car1_move
    new_p_1619_Mark = 0; // Place Car2_move
    new_p_1633_Mark = 0; // Place Car3_move
    new_p_1647_Mark = 0; // Place Car1_ready
    new_p_1661_Mark = 0; // Place Car2_ready
    new_p_1675_Mark = 0; // Place Car3_ready
    new_p_1703_Mark = 0; // Place Car2_at_end
    new_p_1717_Mark = 0; // Place Car1_at_end
    new_p_1731_Mark = 0; // Place Car1_move_back
    new_p_1745_Mark = 0; // Place Car2_move_back
    new_p_1759_Mark = 0; // Place Car3_move_back
    new_p_2112_Mark = 0; // Place Car3_at_end

    aux_p_1605_Mark = p_1605_Mark; // Place Car1_move
    aux_p_1619_Mark = p_1619_Mark; // Place Car2_move
    aux_p_1633_Mark = p_1633_Mark; // Place Car3_move
    aux_p_1647_Mark = p_1647_Mark; // Place Car1_ready
    aux_p_1661_Mark = p_1661_Mark; // Place Car2_ready
    aux_p_1675_Mark = p_1675_Mark; // Place Car3_ready
    aux_p_1703_Mark = p_1703_Mark; // Place Car2_at_end
    aux_p_1717_Mark = p_1717_Mark; // Place Car1_at_end
    aux_p_1731_Mark = p_1731_Mark; // Place Car1_move_back
    aux_p_1745_Mark = p_1745_Mark; // Place Car2_move_back
    aux_p_1759_Mark = p_1759_Mark; // Place Car3_move_back
    aux_p_2112_Mark = p_2112_Mark; // Place Car3_at_end

    //Save Output Signals
    last_outSignal_Car3Moves = outSignal_Car3Moves;
    last_outSignal_Car1Moves = outSignal_Car1Moves;
    last_outSignal_Car2Moves = outSignal_Car2Moves;
    last_outSignal_Car2MovesBack = outSignal_Car2MovesBack;
    last_outSignal_Car1MovesBack = outSignal_Car1MovesBack;
    last_outSignal_Car3MovesBack = outSignal_Car3MovesBack;

    // Reset Output Events
    outEvent_Car2Moves = 0;
    outEvent_Car1Moves = 0;
    outEvent_Car3Moves = 0;
    outEvent_Car1MovesBack = 0;
    outEvent_Car2MovesBack = 0;
    outEvent_Car3MovesBack = 0;
```

```
Analyse_Input_Events();


// Transition 1443 ( GO )
if(aux_p_1647_Mark >= 1 && aux_p_1661_Mark >= 1 && aux_p_1675_Mark >= 1 && inEvent_GO
                                == 1)
 {
   aux_p_1647_Mark = aux_p_1647_Mark - 1;
   aux_p_1661_Mark = aux_p_1661_Mark - 1;
   aux_p_1675_Mark = aux_p_1675_Mark - 1;
   new_p_1605_Mark = new_p_1605_Mark + 1;
   new_p_1619_Mark = new_p_1619_Mark + 1;
   new_p_1633_Mark = new_p_1633_Mark + 1;
}

// Transition 1461 ( B1 )
if(aux_p_1605_Mark >= 1 && inEvent_GATEc1 == 1)
 {
   aux_p_1605_Mark = aux_p_1605_Mark - 1;
   new_p_1717_Mark = new_p_1717_Mark + 1;
}

// Transition 1479 ( A1 )
if(aux_p_1731_Mark >= 1 && inEvent_GATEc1 == 1)
 {
   aux_p_1731_Mark = aux_p_1731_Mark - 1;
   new_p_1647_Mark = new_p_1647_Mark + 1;
}

// Transition 1515 ( B2 )
if(aux_p_1619_Mark >= 1 && inEvent_GATEc2 == 1)
 {
   aux_p_1619_Mark = aux_p_1619_Mark - 1;
   new_p_1703_Mark = new_p_1703_Mark + 1;
}

// Transition 1551 ( B3 )
if(aux_p_1633_Mark >= 1 && inEvent_GATEc3 == 1)
 {
   aux_p_1633_Mark = aux_p_1633_Mark - 1;
   new_p_2112_Mark = new_p_2112_Mark + 1;
}

// Transition 1569 ( A3 )
if(aux_p_1759_Mark >= 1 && inEvent_GATEc3 == 1)
 {
   aux_p_1759_Mark = aux_p_1759_Mark - 1;
   new_p_1675_Mark = new_p_1675_Mark + 1;
}

// Transition 1587 ( A2 )
if(aux_p_1745_Mark >= 1 && inEvent_GATEc2 == 1)
 {
```

```
   aux_p_1745_Mark = aux_p_1745_Mark - 1;
   new_p_1661_Mark = new_p_1661_Mark + 1;
}

// Transition 1805 ( BACK )
if(aux_p_1717_Mark >= 1 && aux_p_1703_Mark >= 1 && aux_p_2112_Mark >= 1 &&
                          inEvent_BACK == 1)
 {
   aux_p_1717_Mark = aux_p_1717_Mark - 1;
   aux_p_1703_Mark = aux_p_1703_Mark - 1;
   aux_p_2112_Mark = aux_p_2112_Mark - 1;
   new_p_1731_Mark = new_p_1731_Mark + 1;
   new_p_1745_Mark = new_p_1745_Mark + 1;
   new_p_1759_Mark = new_p_1759_Mark + 1;
}

// Actualize Marks
p_1605_Mark = aux_p_1605_Mark + new_p_1605_Mark;
p_1619_Mark = aux_p_1619_Mark + new_p_1619_Mark;
p_1633_Mark = aux_p_1633_Mark + new_p_1633_Mark;
p_1647_Mark = aux_p_1647_Mark + new_p_1647_Mark;
p_1661_Mark = aux_p_1661_Mark + new_p_1661_Mark;
p_1675_Mark = aux_p_1675_Mark + new_p_1675_Mark;
p_1703_Mark = aux_p_1703_Mark + new_p_1703_Mark;
p_1717_Mark = aux_p_1717_Mark + new_p_1717_Mark;
p_1731_Mark = aux_p_1731_Mark + new_p_1731_Mark;
p_1745_Mark = aux_p_1745_Mark + new_p_1745_Mark;
p_1759_Mark = aux_p_1759_Mark + new_p_1759_Mark;
p_2112_Mark = aux_p_2112_Mark + new_p_2112_Mark;

//Save Input Signals
last_inSignal_GO = inSignal_GO;
last_inSignal_BACK = inSignal_BACK;
last_inSignal_GATEc1 = inSignal_GATEc1;
last_inSignal_GATEc2 = inSignal_GATEc2;
last_inSignal_GATEc3 = inSignal_GATEc3;

// Reset Input Events
inEvent_GO = 0;
inEvent_BACK = 0;
inEvent_GATEc1 = 0;
inEvent_GATEc2 = 0;
inEvent_GATEc3 = 0;

outSignal_Car3Moves = 0;
outSignal_Car1Moves = 0;
outSignal_Car2Moves = 0;
outSignal_Car2MovesBack = 0;
outSignal_Car1MovesBack = 0;
outSignal_Car3MovesBack = 0;

   if(p_1605_Mark >= 1)
     {
       if(outSignal_Car1Moves < 1)
```

```
            outSignal_Car1Moves = 1;
        }
        if(p_1619_Mark >= 1)
         {
           if(outSignal_Car2Moves < 1)
            outSignal_Car2Moves = 1;
        }
        if(p_1633_Mark >= 1)
         {
           if(outSignal_Car3Moves < 1)
            outSignal_Car3Moves = 1;
        }
        if(p_1731_Mark >= 1)
         {
           if(outSignal_Car1MovesBack < 1)
            outSignal_Car1MovesBack = 1;
        }
        if(p_1745_Mark >= 1)
         {
           if(outSignal_Car2MovesBack < 1)
            outSignal_Car2MovesBack = 1;
        }
        if(p_1759_Mark >= 1)
         {
           if(outSignal_Car3MovesBack < 1)
            outSignal_Car3MovesBack = 1;
        }

    Analyse_Output_Events();

    return 0;
}
```

ANEXO 3 – FICHEIROS GERADOS PARA O CARRO 1

MAIN.C

```
#include <string.h>
#include "ioptc.h"

//----------------Input Signals Values----------------------
char inSignal_GO;char last_inSignal_GO;
char inSignal_GATEc1;char last_inSignal_GATEc1;
char inSignal_BACK;char last_inSignal_BACK;
//------------------------------------------------------------

//----------------Output Signals Values----------------------
char outSignal_Car1MovesBack;char last_outSignal_Car1MovesBack;
char outSignal_Car1Moves;char last_outSignal_Car1Moves;
//------------------------------------------------------------

//----------------Input Events Values----------------------
char inEvent_GO;
char inevent1587;
char inevent1569;
char inEvent_GATEc1;
char inEvent_BACK;
char inevent1551;
char inevent1515;
//------------------------------------------------------------

//----------------Output Events Values----------------------
char outevent1443;
char outevent1805;
//------------------------------------------------------------

//----------------------Place Marks--------------------------
char p_18052684_Mark;char aux_p_18052684_Mark;char new_p_18052684_Mark;
char p_18052698_Mark;char aux_p_18052698_Mark;char new_p_18052698_Mark;
char p_18052712_Mark;char aux_p_18052712_Mark;char new_p_18052712_Mark;
char p_18052726_Mark;char aux_p_18052726_Mark;char new_p_18052726_Mark;
char p_18053438_Mark;char aux_p_18053438_Mark;char new_p_18053438_Mark;
char p_18053452_Mark;char aux_p_18053452_Mark;char new_p_18053452_Mark;
char p_18053466_Mark;char aux_p_18053466_Mark;char new_p_18053466_Mark;
char p_18053480_Mark;char aux_p_18053480_Mark;char new_p_18053480_Mark;
//------------------------------------------------------------


int main()
{
  start();
```

```
  //insert your code here...

  //Set Input Signals
  //Set Input Events

  run();

  //Get Output Signals
  //Get Output Events

  //insert your code here...

  return 0;
}
```

FUNCTIONS.H

```
#include <string.h>
#include <stdio.h>
#include <stdlib.h>

//----------------Input Signals Values----------------------
extern char inSignal_GO;extern char last_inSignal_GO;
extern char inSignal_GATEc1;extern char last_inSignal_GATEc1;
extern char inSignal_BACK;extern char last_inSignal_BACK;
//----------------------------------------------------------

//----------------Output Signals Values---------------------
extern char outSignal_Car1MovesBack;extern char last_outSignal_Car1MovesBack;
extern char outSignal_Car1Moves;extern char last_outSignal_Car1Moves;
//----------------------------------------------------------

//----------------Input Events Values-----------------------
extern char inEvent_GO;
extern char inevent1587;
extern char inevent1569;
extern char inEvent_GATEc1;
extern char inEvent_BACK;
extern char inevent1551;
extern char inevent1515;
//----------------------------------------------------------

//----------------Output Events Values----------------------
extern char outevent1443;
extern char outevent1805;
//----------------------------------------------------------

//----------------Place Marks-------------------
```

```
extern	char	p_18052684_Mark;extern	char	aux_p_18052684_Mark;extern	char
		new_p_18052684_Mark;
extern	char	p_18052698_Mark;extern	char	aux_p_18052698_Mark;extern	char
		new_p_18052698_Mark;
extern	char	p_18052712_Mark;extern	char	aux_p_18052712_Mark;extern	char
		new_p_18052712_Mark;
extern	char	p_18052726_Mark;extern	char	aux_p_18052726_Mark;extern	char
		new_p_18052726_Mark;
extern	char	p_18053438_Mark;extern	char	aux_p_18053438_Mark;extern	char
		new_p_18053438_Mark;
extern	char	p_18053452_Mark;extern	char	aux_p_18053452_Mark;extern	char
		new_p_18053452_Mark;
extern	char	p_18053466_Mark;extern	char	aux_p_18053466_Mark;extern	char
		new_p_18053466_Mark;
extern	char	p_18053480_Mark;extern	char	aux_p_18053480_Mark;extern	char
		new_p_18053480_Mark;
//-------------------------------------------------------


//Input Signals
void Set_inSignal_GO (char val);
void Set_inSignal_GATEc1 (char val);
void Set_inSignal_BACK (char val);


//Output Signal
char Get_outSignal_Car1MovesBack (void);
char Get_outSignal_Car1Moves (void);


//Input Event
void Set_inEvent_GO (char val);
void Set_inevent1587 (char val);
void Set_inevent1569 (char val);
void Set_inEvent_GATEc1 (char val);
void Set_inEvent_BACK (char val);
void Set_inevent1551 (char val);
void Set_inevent1515 (char val);


//Output Event
char Get_outevent1443 (void);
char Get_outevent1805 (void);

void Analyse_Input_Events(void);
void Analyse_Output_Events(void);
```

FUNCTIONS.C

```
#include "functions.h"

//Input Signal
void Set_inSignal_GO (char val)
{
  last_inSignal_GO = inSignal_GO;
  inSignal_GO = val;
}

//Input Signal
void Set_inSignal_GATEc1 (char val)
{
  last_inSignal_GATEc1 = inSignal_GATEc1;
  inSignal_GATEc1 = val;
}

//Input Signal
void Set_inSignal_BACK (char val)
{
  last_inSignal_BACK = inSignal_BACK;
  inSignal_BACK = val;
}


//Output Signal
char Get_outSignal_Car1MovesBack (void)
{
  return outSignal_Car1MovesBack;
}

//Output Signal
char Get_outSignal_Car1Moves (void)
{
  return outSignal_Car1Moves;
}


//Input Event
void Set_inEvent_GO (char val)
{
  inEvent_GO = val;
}

//Input Event
void Set_inevent1587 (char val)
{
  inevent1587 = val;
}

//Input Event
```

```
void Set_inevent1569 (char val)
{
  inevent1569 = val;
}

//Input Event
void Set_inEvent_GATEc1 (char val)
{
  inEvent_GATEc1 = val;
}

//Input Event
void Set_inEvent_BACK (char val)
{
  inEvent_BACK = val;
}

//Input Event
void Set_inevent1551 (char val)
{
  inevent1551 = val;
}

//Input Event
void Set_inevent1515 (char val)
{
  inevent1515 = val;
}


//Output Event
char Get_outevent1443 (void)
{
  return outevent1443;
}

//Output Event
char Get_outevent1805 (void)
{
  return outevent1805;
}

void Analyse_Input_Events(void)
{
  if(inSignal_GO == 1 && last_inSignal_GO == 0)
     inEvent_GO = 1;
  if(inSignal_GATEc1 == 1 && last_inSignal_GATEc1 == 0)
     inEvent_GATEc1 = 1;
  if(inSignal_BACK == 1 && last_inSignal_BACK == 0)
     inEvent_BACK = 1;
}
void Analyse_Output_Events(void)
{
}
```

IOPTC.H

```
#include <string.h>
#include <stdio.h>
#include <stdlib.h>
#include "functions.h"


//----------------Input Signals Values----------------------
extern char inSignal_GO;extern char last_inSignal_GO;
extern char inSignal_GATEc1;extern char last_inSignal_GATEc1;
extern char inSignal_BACK;extern char last_inSignal_BACK;
//------------------------------------------------------------

//----------------Output Signals Values----------------------
extern char outSignal_Car1MovesBack;extern char last_outSignal_Car1MovesBack;
extern char outSignal_Car1Moves;extern char last_outSignal_Car1Moves;
//------------------------------------------------------------

//----------------Input Events Values----------------------
extern char inEvent_GO;
extern char inevent1587;
extern char inevent1569;
extern char inEvent_GATEc1;
extern char inEvent_BACK;
extern char inevent1551;
extern char inevent1515;
//------------------------------------------------------------

//----------------Output Events Values----------------------
extern char outevent1443;
extern char outevent1805;
//------------------------------------------------------------


//----------------Place Marks----------------------
extern char p_18052684_Mark;extern char aux_p_18052684_Mark;extern char
                                    new_p_18052684_Mark;
extern char p_18052698_Mark;extern char aux_p_18052698_Mark;extern char
                                    new_p_18052698_Mark;
extern char p_18052712_Mark;extern char aux_p_18052712_Mark;extern char
                                    new_p_18052712_Mark;
extern char p_18052726_Mark;extern char aux_p_18052726_Mark;extern char
                                    new_p_18052726_Mark;
extern char p_18053438_Mark;extern char aux_p_18053438_Mark;extern char
                                    new_p_18053438_Mark;
extern char p_18053452_Mark;extern char aux_p_18053452_Mark;extern char
                                    new_p_18053452_Mark;
extern char p_18053466_Mark;extern char aux_p_18053466_Mark;extern char
                                    new_p_18053466_Mark;
extern char p_18053480_Mark;extern char aux_p_18053480_Mark;extern char
                                    new_p_18053480_Mark;
```

```
//-------------------------------------------------------

char start(void);
char run(void);
```

IOPTC.C

```
#include "ioptc.h"

char start(void)
{
//----------------Init Input Signals Values---------------------
     inSignal_GO = 0;
     inSignal_GATEc1 = 0;
     inSignal_BACK = 0;
//-------------------------------------------------------

//----------------Init Output Signals Values---------------------
     outSignal_Car1MovesBack = 0;
     outSignal_Car1Moves = 0;
//-------------------------------------------------------

//----------------Init Input Events Values---------------------
     inEvent_GO = 0;
     inevent1587 = 0;
     inevent1569 = 0;
     inEvent_GATEc1 = 0;
     inEvent_BACK = 0;
     inevent1551 = 0;
     inevent1515 = 0;
//-------------------------------------------------------

//----------------Init Output Events Values---------------------
     outevent1443 = 0;
     outevent1805 = 0;
//-------------------------------------------------------

//----------------Init Place Marks---------------------
     p_18052684_Mark = 1; // Place Car3_ready_m
     p_18052698_Mark = 1; // Place Car1_ready
     p_18052712_Mark = 0; // Place Car1_move_back
     p_18052726_Mark = 1; // Place Car2_ready_m
     p_18053438_Mark = 0; // Place Car1_move
     p_18053452_Mark = 0; // Place Car2_at_end_m
     p_18053466_Mark = 0; // Place Car3_at_end_m
     p_18053480_Mark = 0; // Place Car1_at_end
//-------------------------------------------------------

     return 0;
```

```
}

char run(void)
{

    new_p_18052684_Mark = 0; // Place Car3_ready_m
    new_p_18052698_Mark = 0; // Place Car1_ready
    new_p_18052712_Mark = 0; // Place Car1_move_back
    new_p_18052726_Mark = 0; // Place Car2_ready_m
    new_p_18053438_Mark = 0; // Place Car1_move
    new_p_18053452_Mark = 0; // Place Car2_at_end_m
    new_p_18053466_Mark = 0; // Place Car3_at_end_m
    new_p_18053480_Mark = 0; // Place Car1_at_end

    aux_p_18052684_Mark = p_18052684_Mark; // Place Car3_ready_m
    aux_p_18052698_Mark = p_18052698_Mark; // Place Car1_ready
    aux_p_18052712_Mark = p_18052712_Mark; // Place Car1_move_back
    aux_p_18052726_Mark = p_18052726_Mark; // Place Car2_ready_m
    aux_p_18053438_Mark = p_18053438_Mark; // Place Car1_move
    aux_p_18053452_Mark = p_18053452_Mark; // Place Car2_at_end_m
    aux_p_18053466_Mark = p_18053466_Mark; // Place Car3_at_end_m
    aux_p_18053480_Mark = p_18053480_Mark; // Place Car1_at_end

    //Save Output Signals
    last_outSignal_Car1MovesBack = outSignal_Car1MovesBack;
    last_outSignal_Car1Moves = outSignal_Car1Moves;

    // Reset Output Events
    outevent1443 = 0;
    outevent1805 = 0;

    Analyse_Input_Events();


    // Transition 18052740 ( A2_m )
    if(inevent1587 == 1)
     {
       new_p_18052726_Mark = new_p_18052726_Mark + 1;
    }

    // Transition 18052758 ( A3_m )
    if(inevent1569 == 1)
     {
       new_p_18052684_Mark = new_p_18052684_Mark + 1;
    }

    // Transition 18052776 ( A1 )
    if(aux_p_18052712_Mark >= 1 && inEvent_GATEc1 == 1)
     {
       aux_p_18052712_Mark = aux_p_18052712_Mark - 1;
       new_p_18052698_Mark = new_p_18052698_Mark + 1;
    }

    // Transition 18052794 ( BACK_m2 )
```

```
if(aux_p_18053480_Mark >= 1 && aux_p_18053452_Mark >= 1 && aux_p_18053466_Mark >= 1
                                && inEvent_BACK == 1)
 {
   aux_p_18053480_Mark = aux_p_18053480_Mark - 1;
   aux_p_18053452_Mark = aux_p_18053452_Mark - 1;
   aux_p_18053466_Mark = aux_p_18053466_Mark - 1;
   new_p_18052712_Mark = new_p_18052712_Mark + 1;
   outevent1805 = 1;
}

// Transition 18052812 ( GO )
if(aux_p_18052726_Mark >= 1 && aux_p_18052698_Mark >= 1 && aux_p_18052684_Mark >= 1
                                && inEvent_GO == 1)
 {
   aux_p_18052726_Mark = aux_p_18052726_Mark - 1;
   aux_p_18052698_Mark = aux_p_18052698_Mark - 1;
   aux_p_18052684_Mark = aux_p_18052684_Mark - 1;
   new_p_18053438_Mark = new_p_18053438_Mark + 1;
   outevent1443 = 1;
}

// Transition 18053494 ( B1 )
if(aux_p_18053438_Mark >= 1 && inEvent_GATEc1 == 1)
 {
   aux_p_18053438_Mark = aux_p_18053438_Mark - 1;
   new_p_18053480_Mark = new_p_18053480_Mark + 1;
}

// Transition 18053530 ( B3_m )
if(inevent1551 == 1)
 {
   new_p_18053466_Mark = new_p_18053466_Mark + 1;
}

// Transition 18053566 ( B2_m )
if(inevent1515 == 1)
 {
   new_p_18053452_Mark = new_p_18053452_Mark + 1;
}

// Actualize Marks
p_18052684_Mark = aux_p_18052684_Mark + new_p_18052684_Mark;
p_18052698_Mark = aux_p_18052698_Mark + new_p_18052698_Mark;
p_18052712_Mark = aux_p_18052712_Mark + new_p_18052712_Mark;
p_18052726_Mark = aux_p_18052726_Mark + new_p_18052726_Mark;
p_18053438_Mark = aux_p_18053438_Mark + new_p_18053438_Mark;
p_18053452_Mark = aux_p_18053452_Mark + new_p_18053452_Mark;
p_18053466_Mark = aux_p_18053466_Mark + new_p_18053466_Mark;
p_18053480_Mark = aux_p_18053480_Mark + new_p_18053480_Mark;

//Save Input Signals
last_inSignal_GO = inSignal_GO;
last_inSignal_GATEc1 = inSignal_GATEc1;
last_inSignal_BACK = inSignal_BACK;
```

```
    // Reset Input Events
    inEvent_GO = 0;
    inevent1587 = 0;
    inevent1569 = 0;
    inEvent_GATEc1 = 0;
    inEvent_BACK = 0;
    inevent1551 = 0;
    inevent1515 = 0;

    outSignal_Car1MovesBack = 0;
    outSignal_Car1Moves = 0;

       if(p_18052712_Mark >= 1)
        {
          if(outSignal_Car1MovesBack < 1)
            outSignal_Car1MovesBack = 1;
       }
       if(p_18053438_Mark >= 1)
        {
          if(outSignal_Car1Moves < 1)
            outSignal_Car1Moves = 1;
       }

    Analyse_Output_Events();

    return 0;
}
```

ANEXO 4 – FICHEIROS GERADOS PARA O CARRO 3

MAIN.C

```
#include <string.h>
#include "ioptc.h"

//----------------Input Signals Values---------------------
char inSignal_GATEc3;char last_inSignal_GATEc3;
//---------------------------------------------------------

//----------------Output Signals Values---------------------
char outSignal_Car3Moves;char last_outSignal_Car3Moves;
char outSignal_Car3MovesBack;char last_outSignal_Car3MovesBack;
//---------------------------------------------------------

//----------------Input Events Values---------------------
char inEvent_GATEc3;
char inevent1443;
char inevent1805;
//---------------------------------------------------------

//----------------Output Events Values---------------------
char outevent1551;
char outevent1569;
//---------------------------------------------------------

//----------------------Place Marks--------------------------
char p_18057479_Mark;char aux_p_18057479_Mark;char new_p_18057479_Mark;
char p_18057493_Mark;char aux_p_18057493_Mark;char new_p_18057493_Mark;
char p_18057945_Mark;char aux_p_18057945_Mark;char new_p_18057945_Mark;
char p_18057959_Mark;char aux_p_18057959_Mark;char new_p_18057959_Mark;
//---------------------------------------------------------


int main()
{
  start();

  //insert your code here...

  //Set Input Signals
  //Set Input Events

  run();

  //Get Output Signals
  //Get Output Events
```

```
  //insert your code here...

  return 0;
}
```

FUNCTIONS.H

```
#include <string.h>
#include <stdio.h>
#include <stdlib.h>

//----------------Input Signals Values----------------------
extern char inSignal_GATEc3;extern char last_inSignal_GATEc3;
//----------------------------------------------------------

//----------------Output Signals Values---------------------
extern char outSignal_Car3Moves;extern char last_outSignal_Car3Moves;
extern char outSignal_Car3MovesBack;extern char last_outSignal_Car3MovesBack;
//----------------------------------------------------------

//----------------Input Events Values-----------------------
extern char inEvent_GATEc3;
extern char inevent1443;
extern char inevent1805;
//----------------------------------------------------------

//----------------Output Events Values----------------------
extern char outevent1551;
extern char outevent1569;
//----------------------------------------------------------

//----------------Place Marks-------------------------
extern char p_18057479_Mark;extern char aux_p_18057479_Mark;extern char
                                    new_p_18057479_Mark;
extern char p_18057493_Mark;extern char aux_p_18057493_Mark;extern char
                                    new_p_18057493_Mark;
extern char p_18057945_Mark;extern char aux_p_18057945_Mark;extern char
                                    new_p_18057945_Mark;
extern char p_18057959_Mark;extern char aux_p_18057959_Mark;extern char
                                    new_p_18057959_Mark;
//----------------------------------------------------------



//Input Signals
void Set_inSignal_GATEc3 (char val);
```

```
//Output Signal
char Get_outSignal_Car3Moves (void);
char Get_outSignal_Car3MovesBack (void);


//Input Event
void Set_inEvent_GATEc3 (char val);
void Set_inevent1443 (char val);
void Set_inevent1805 (char val);


//Output Event
char Get_outevent1551 (void);
char Get_outevent1569 (void);

void Analyse_Input_Events(void);
void Analyse_Output_Events(void);
```

FUNCTIONS.C

```
#include "functions.h"

//Input Signal
void Set_inSignal_GATEc3 (char val)
{
  last_inSignal_GATEc3 = inSignal_GATEc3;
  inSignal_GATEc3 = val;
}


//Output Signal
char Get_outSignal_Car3Moves (void)
{
  return outSignal_Car3Moves;
}

//Output Signal
char Get_outSignal_Car3MovesBack (void)
{
  return outSignal_Car3MovesBack;
}


//Input Event
void Set_inEvent_GATEc3 (char val)
{
  inEvent_GATEc3 = val;
}
```

```
//Input Event
void Set_inevent1443 (char val)
{
  inevent1443 = val;
}

//Input Event
void Set_inevent1805 (char val)
{
  inevent1805 = val;
}


//Output Event
char Get_outevent1551 (void)
{
  return outevent1551;
}

//Output Event
char Get_outevent1569 (void)
{
  return outevent1569;
}

void Analyse_Input_Events(void)
{
  if(inSignal_GATEc3 == 1 && last_inSignal_GATEc3 == 0)
     inEvent_GATEc3 = 1;
}

void Analyse_Output_Events(void)
{
}
```

IOPTC.H

```
#include <string.h>
#include <stdio.h>
#include <stdlib.h>
#include "functions.h"


//----------------Input Signals Values----------------------
extern char inSignal_GATEc3;extern char last_inSignal_GATEc3;
//----------------------------------------------------------

//----------------Output Signals Values---------------------
extern char outSignal_Car3Moves;extern char last_outSignal_Car3Moves;
```

```
extern char outSignal_Car3MovesBack;extern char last_outSignal_Car3MovesBack;
//------------------------------------------------------

//----------------Input Events Values---------------------
extern char inEvent_GATEc3;
extern char inevent1443;
extern char inevent1805;
//------------------------------------------------------

//----------------Output Events Values---------------------
extern char outevent1551;
extern char outevent1569;
//------------------------------------------------------


//----------------Place Marks---------------------
extern char p_18057479_Mark;extern char aux_p_18057479_Mark;extern char
                                    new_p_18057479_Mark;
extern char p_18057493_Mark;extern char aux_p_18057493_Mark;extern char
                                    new_p_18057493_Mark;
extern char p_18057945_Mark;extern char aux_p_18057945_Mark;extern char
                                    new_p_18057945_Mark;
extern char p_18057959_Mark;extern char aux_p_18057959_Mark;extern char
                                    new_p_18057959_Mark;
//------------------------------------------------------

char start(void);
char run(void);
```

IOPTC.C

```
#include "ioptc.h"

char start(void)
{
//----------------Init Input Signals Values---------------------
    inSignal_GATEc3 = 0;
//------------------------------------------------------

//----------------Init Output Signals Values---------------------
    outSignal_Car3Moves = 0;
    outSignal_Car3MovesBack = 0;
//------------------------------------------------------

//----------------Init Input Events Values---------------------
    inEvent_GATEc3 = 0;
    inevent1443 = 0;
    inevent1805 = 0;
//------------------------------------------------------
```

```
//----------------Init Output Events Values----------------------
    outevent1551 = 0;
    outevent1569 = 0;
//-----------------------------------------------------------

//----------------Init Place Marks---------------------
    p_18057479_Mark = 0; // Place Car3_move
    p_18057493_Mark = 0; // Place Car3_at_end
    p_18057945_Mark = 0; // Place Car3_move_back
    p_18057959_Mark = 1; // Place Car3_ready
//-----------------------------------------------------------

    return 0;
}

char run(void)
{

    new_p_18057479_Mark = 0; // Place Car3_move
    new_p_18057493_Mark = 0; // Place Car3_at_end
    new_p_18057945_Mark = 0; // Place Car3_move_back
    new_p_18057959_Mark = 0; // Place Car3_ready

    aux_p_18057479_Mark = p_18057479_Mark; // Place Car3_move
    aux_p_18057493_Mark = p_18057493_Mark; // Place Car3_at_end
    aux_p_18057945_Mark = p_18057945_Mark; // Place Car3_move_back
    aux_p_18057959_Mark = p_18057959_Mark; // Place Car3_ready

    //Save Output Signals
    last_outSignal_Car3Moves = outSignal_Car3Moves;
    last_outSignal_Car3MovesBack = outSignal_Car3MovesBack;

    // Reset Output Events
    outevent1551 = 0;
    outevent1569 = 0;

    Analyse_Input_Events();


    // Transition 18057543 ( B3 )
    if(aux_p_18057479_Mark >= 1 && inEvent_GATEc3 == 1)
     {
       aux_p_18057479_Mark = aux_p_18057479_Mark - 1;
       new_p_18057493_Mark = new_p_18057493_Mark + 1;
       outevent1551 = 1;
    }

    // Transition 18057973 ( GO_m5 )
    if(aux_p_18057959_Mark >= 1 && inevent1443 == 1)
     {
       aux_p_18057959_Mark = aux_p_18057959_Mark - 1;
       new_p_18057479_Mark = new_p_18057479_Mark + 1;
    }
```

```
    // Transition 18057991 ( A3 )
    if(aux_p_18057945_Mark >= 1 && inEvent_GATEc3 == 1)
     {
       aux_p_18057945_Mark = aux_p_18057945_Mark - 1;
       new_p_18057959_Mark = new_p_18057959_Mark + 1;
       outevent1569 = 1;
     }

    // Transition 18058009 ( BACK_m4 )
    if(aux_p_18057493_Mark >= 1 && inevent1805 == 1)
     {
       aux_p_18057493_Mark = aux_p_18057493_Mark - 1;
       new_p_18057945_Mark = new_p_18057945_Mark + 1;
     }

    // Actualize Marks
    p_18057479_Mark = aux_p_18057479_Mark + new_p_18057479_Mark;
    p_18057493_Mark = aux_p_18057493_Mark + new_p_18057493_Mark;
    p_18057945_Mark = aux_p_18057945_Mark + new_p_18057945_Mark;
    p_18057959_Mark = aux_p_18057959_Mark + new_p_18057959_Mark;

    //Save Input Signals
    last_inSignal_GATEc3 = inSignal_GATEc3;

    // Reset Input Events
    inEvent_GATEc3 = 0;
    inevent1443 = 0;
    inevent1805 = 0;

    outSignal_Car3Moves = 0;
    outSignal_Car3MovesBack = 0;

       if(p_18057479_Mark >= 1)
         {
           if(outSignal_Car3Moves < 1)
             outSignal_Car3Moves = 1;
       }
       if(p_18057945_Mark >= 1)
         {
           if(outSignal_Car3MovesBack < 1)
             outSignal_Car3MovesBack = 1;
       }

    Analyse_Output_Events();

    return 0;
}
```